UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Jernej Hartman

Koncepti izdelave mobilnih iger v operacijskem sistemu Android

DIPLOMSKO DELO
VISOKOŠOLSKI STROKOVNI ŠTUDIJSKI PROGRAM PRVE STOPNJE
RAČUNALNIŠTVO IN INFORMATIKA

Mentor: doc. dr. Tomaž Dobravec

Ljubljana, 2011

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

Tržaška 25
1000 Ljubljana, Slovenija
telefon: 01 476 84 11
faks: 01 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00093/2011
Datum: 04.04.2011

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **JERNEJ HARTMAN**

Naslov: **KONCEPTI IZDELAVE MOBILNIH IGER V OPERACIJSKEM SISTEMU ANDROID**

THE CONCEPTS OF DEVELOPING MOBILE GAMES FOR THE OPERATING SYSTEM ANDROID

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

Predstavite pojem mobilne igre ter koncept, ki se uporablja pri njeni izdelavi. Osredotočite se na podobnosti in razlike pri izdelavi mobilne igre v različnih operacijskih sistemih ter opišite prijeme s katerimi hitrost izvajanja mobilne igre prilagajamo zmogljivosti mobilne naprave, na kateri se igra izvaja. Predstavite operacijski sistem Android in opišite postopek izdelave mobilne igre zanj. Izdelajte mobilno igro za operacijski sistem Android in predstavite težave in posebnosti, s katerimi se boste pri izdelavi srečali.

Mentor:
doc. dr. Tomaž Dobravec

Dekan:
prof. dr. Nikolaj Zimic

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

I Z J A V A O A V T O R S T V U

diplomskega dela

Spodaj podpisani Jernej Hartman,

z vpisno številko 63060483,

sem avtor diplomskega dela z naslovom:

Koncepti izdelave mobilnih iger v operacijskem sistemu Android

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom

  doc. dr. Tomaža Dobravca

- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela

- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki »Dela FRI«.

V Ljubljani, dne 01.07.2011 Podpis avtorja:

Zahvala

Zahvaljujem se mentorju, doc. dr. Tomažu Dobravcu za vodenje in strokovno pomoč pri izdelavi diplomskega dela.

Kazalo

Seznam uporabljenih kratic in simbolov

2D – Two Dimensional; dvo dimenzionalni

3D - Three dimensional; tri dimenzionalni

ADT – Android Development Tools; razvojna orodja za Androida

AI – Artificial Intelligence; umetna inteligenca

ANR – Application Not Responding; aplikacija se ne odziva

API – Application Programming Interface; aplikativni programski vmesnik

AVD – Android Virtual Device; navidezna naprava Android

CDMA – Code Division Multiple Access; hkraten dostop z deljenjem kode

CPU – Central Processing Unit; Centralno procesna enota

DPI – Dots Per Inch; število točk na inč

EDGE – Enhanced Data rates for GSM Evolution; izboljšane vrednosti hitrosti prenosa za globalni napredek

EV-DO – Evolution-Data Optimized; evolucijsko-podatkovno optimiziran

FPS – Frames Per Second; sličic na sekundo

FPU – Floating Point Unit; enota za plavajočo vejico

GPS – Global Positioning System; Sistem globalnega določanja položaja

GPU – Graphics processing unit; grafična procesna enota

GSM – Global System for Mobile communications; globalni sistem za mobilno komunikacijo

IDE – Integrated Development Environment; vgrajeno razvojno okolje

IDEN – Integrated Digital Enhanced Network ; integrirano izboljšano digitalno omrežje

J2ME – Java 2 Micro Edition; Java 2 mikro verzija

LIFO – Last In First Out; Zadnji notri, prvi ven

LTE – 3GPP Long Term Evolution; dolgoročna 3GPP evolucija

NFC – Near Field Communication; komunikacija bližnjih polj

PX – Pixel; točka

SDK – Software Development Kit; paket za razvoj programske opreme

SMS – Short Message Service; storitev za kratka sporočila

SQL – Structured Query Language; sestavljen poizvedni jezik

UMTS – Universal Mobile Telecommunications System; univerzalni mobilni telekomunikacijski sistem

UPS – Updates Per Second; posodobitev na sekundo

VM – Virtual Machine; navidezni stroj

WiMAX – Worldwide Interoperability for Microwave Access; svetovna interoperatibilnost mikrovalovnega dostopa

XML – eXtensible Markup Language; razširljiv označevalni jezik

Povzetek

Diplomsko delo opisuje koncepte izdelave mobilnih iger v operacijskem sistemu Android. Delo se osredotoča na področje mobilnih iger v povezavi z mobilnimi operacijskimi sistemi in mobilnimi napravami. Nadaljuje se s predstavitvijo samega koncepta delovanja igre in primerjavo različnih metod implementacije igre. Diplomsko delo v nadaljevanju prikazuje in opisuje ciljni operacijski sistem – Android. Opisana je njegova arhitektura in delovanje aplikacije znotraj operacijskega sistema. Sledi praktični del diplomske naloge, ki predstavlja igro Creativity, ki je bila razvita za prikaz konceptov za to diplomsko delo. Igra in njene zahteve so na kratko predstavljene, v nadaljevanju pa je izpostavljen razvoj različnih komponent igre, ki so bile potrebne za delovanje. V zaključku so predstavljene sklepne ugotovitve oziroma ideje za nadaljno razširitev igre.

Ključne besede:

Operacijski sistem Android, mobilne igre, koncepti iger, mobilna aplikacija

Abstract

The diploma thesis describes the concepts behind developing mobile games for the Android operating system. The thesis first covers mobile games and everything related to them, from mobile operating systems to mobile devices. It also describes how mobile games work and compares different concepts regarding their implementation. It then presents and covers the target mobile operating system of this thesis – the Android – describing its architecture and the operation of applications using this operating system. This is followed by the practical part of the thesis which comprises the mobile game called Creativity I developed which represents the theory behind how games work. The game introduces the minimum requirements for the game to function properly and describes the development of the various required components. The conclusion presents findings and ideas for the continued expansion of the game.

Key words:

Operating system Android, mobile games, game concepts, mobile application

1 Uvod

V današnjem času si je težko predstavljati življenje brez računalnika. Ponudba osebnih računalnikov je vedno večja, njihove komponente in poraba energije vedno manjše, hitrost pa se konstano izboljšuje. Napredek tehnologije je pripeljal do razvoja mobilnih računalnikov oziroma tako imenovanih pametnih mobilnih naprav, te pa so že tako majhne, da jih lahko nosimo s seboj v žepu. Sprva so bile mobilne naprave namenjene le zvočnemu komuniciranju, danes pa se zaradi vgrajene strojne opreme, kot je na primer dvojedrni procesor, do 1 GB pomnilnika in ločenega grafičnega procesorja po sposobnosti približujejo namiznim računalnikom. Tako jih sedaj lahko uporabljamo za telefonske in video klice, pošiljanje kratkih sporočil, omogočajo nam povezavo z brezžičnim omrežjem in bluetoothom, lahko zajemamo slike in videe v HD formatu, vgrajen imajo lahko tudi GPS sprejemnik, skratka, možnosti za uporabo mobilnih telefonov je danes zelo veliko.

Z razvojem mobilne tehnologije je prišel tudi razvoj mobilnih operacijskih sistemov in s tem različne aplikacije, ki jih operacijski sistem nudi. Velik del mobilnih aplikacij sestavljajo mobilne igre, katerih se ljudje poslužujejo z namenom krajšanja časa, sprostitve in zabave. Strojna oprema mobilnih naprav je vedno bolj napredna, zato je tudi ponudba iger vedno bolj raznolika.

Večina uporabnikov mobilnih iger se med samo uporabo ne zaveda, kaj vse je potrebno za nemoteno delovanje igre. Vsak premik in animacijo karakterja v igri, vsak pritisk na gumb in njegovo posledico je potrebno pravilno sprogramirati. Na koncu pa je vse to potrebno prilagoditi tako, da brez odstopanj deluje na vseh različnih strojnih opremah in platformah.

V diplomskem delu smo se osredotočili na prikaz konceptov in razvoja mobilne igre v operacijskem sistemu Android. Tako smo sprva skušali predstaviti splošni koncept mobilne igre. Nadalje pa smo opisali tudi koncept mobilnega operacijskega sistema ter potrebnih komponent vsake mobilne igre. Posebej smo izpostavili, kaj je dejansko potrebno za tekoče delovanje iger.

V delu diplome, ki predstavlja delovanje igre, so podrobno opisane tudi različne metode izdelave le-te. S pomočjo programske kode je ponazorjeno delovanje predstavljenih metod, opisali pa smo tudi morebitne težave pri delovanju igre na različni strojni opremi.

V tretjem poglavju smo skušali predstaviti vse, kar je potrebno vedeti o operacijskem sistemu Android, o tem, kako deluje in kako razviti aplikacijo zanj. Opisali smo, kaj

vse je potrebno za razvoj in testiranje aplikacije, in nenazadnje smo opisali tudi vse prvine, ki govorijo o delovanju aplikacije v operacijskem sistemu.

Četrto poglavje vsebuje praktični del diplome, kjer je s pomočjo najbolj optimalne metode, ki je opisana v drugem poglavju, predstavljen razvoj igre Creativity v Androidu. Igra je najprej na kratko opisana in predstavi zahteve s strani uporabnika ter strojne in programske opreme. Temu sledi izdelava klasične igre »pozdravljen, svet«, ki uporablja najpreprostejšo obliko zanke iz drugega poglavja. V nadaljevanju se v zanko vpelje v drugem poglavju predstavljena metoda interpolacije. Temu sledijo predstavitve ovir in težav, ki so se pojavile pri realizaciji te igre.

Diplomsko delo se konča s petim poglavjem, ki predstavi ideje za nadgradjo oziroma posodobitev igre.

2 Mobilne igre

2.1 Splošno o mobilnih igrah

Ko govorimo o mobilnih igrah, govorimo o naprednejši vrsti aplikacije, ki se izvaja na mobilni napravi, bodisi pametnem telefonu bodisi dlančniku ali tabličnem računalniku. Ne vključuje iger, ki se izvajajo na sistemih, na katerih je primarni cilj igranje video iger, kot so prenosni PlayStation, Nintendo DS in podobni [1].

Prva mobilna igra je bila »Kača« (ang. Snake) [2], ki jo je bilo mogoče igrati le na določenih modelih mobilnih telefonov Nokia, in je na trg vstopila leta 1997. Od takrat so mobilniki postali veliko bolj napredni in s tem tudi igre. Postale so bolj strojno zahtevne, kompleksnejše in dobile različne načine interakcije. Prve igre so se odvijale na skromnih velikostih pomnilnikov (56KB) brez grafičnih procesorjev, sedaj pa imajo mobilniki na voljo tudi do 1GB pomnilnika, dvojedrni procesor in ločen grafični procesor. Na voljo imajo tudi različne vrste vhodnih naprav za interakcijo z igro – merilnik pospeškov (ang. accelerometer), zaslon na dotik (ang. touchscreen), virtualna tipkovnica, sledilna krogljica (ang. Trackball) in podobni. Včasih je bila interakcija med napravo in uporabnikom možna le preko fizične tipkovice.

Poznamo več zvrsti iger, kot so na primer strategije, sestavljanke, spominske igre, arkadne igre, prvoosebne strelske igre itd. Igre se ločijo po načinu prikazovanja elementov igre na zaslonu, zornem kotu, ki ga ima uporabnik na igro, in hitrosti igre.

Igra se od navadne aplikacije loči po načinu interakcije z igro, hitrem odzivu in animacijah. V igrah ponavadi simuliramo fizikalne lastnosti realnega sveta, medtem ko v aplikacijah predstavljamo različne podatke.

Bistvena razlika med aplikacijo in igro je v tem, da se pri igri zaslon naprave **osveži** oziroma **ponovno izriše** precej pogosteje kot pri navadni aplikaciji. Pri aplikaciji je ponavadi potrebno zaslon ponovno izrisati le, ko se zgodi neko dejanje (na primer pritisk na gumb), medtem ko je pri igrah potrebno za animacijo lika zaslon ponovno izrisati večkrat na sekundo, da dosežemo učinek tekoče animacije. Vse je seveda odvisno od hitrosti igre.

Igra je lahko prikazana v 2D ali pa v 3D prostoru iz različnega zornega kota uporabnika. Glede na hitrost lahko igre razdelimo v dve skupini:

- Igre s **počasnim osveževanjem** – pri njih se zaslon ne posodablja večkrat na sekundo. mednje na primer sodijo šah, Kača, štiri v vrsto, sestavljanke, spominske igr, itd.

- Igre s **hitrim osveževanjem** – pri teh se mora zaslon osvežiti večkrat na sekundo, da dosežemo različne animacije. Takšne so na primer arkadne, strelske, prvoosebne in podobne igre.

Vsaka igra deluje tako, da izvaja tako imenovano zanko programske kode ali »gameloop«. V tej zanki se posodobi stanje igre glede na interakcijo uporabnika z igro in fizikalne lastnosti simulacije igre in to stanje se izriše na zaslon.

Da bi laže razumeli delovanje igre, si poglejmo preprost primer igre Kača. Tu igralec upravlja kačo, ki se premika po zaslonu in sproti pobira hrano. Zaslon vsebuje tudi zid, ki predstavlja oviro, ki se ji mora kača izogniti. Kačo sestavljajo manjši členi, ki so med sabo povezani. Vsak člen je predstavljen s kvadratom vnaprej določene velikosti in barve. S pobiranjem hrane kača zraste, tako da se doda nov člen. Cilj igre je čim bolj podaljšati kačo, ne da bi se pri tem zaletela v svoj rep ali v zid.

Kača se premakne približno vsakih 600 ms za dolžino enega svojega člena, smer pomika pa določi uporabnik s pomočjo načina interakcije, ki je v tem primeru fizična tipkovnica mobilne naprave.

Ker Kača ne potrebuje gladkih premikov, ni potrebe po hitrem osveževanju, zato spada med igre s počasnim osveževanjem.

Da igra deluje, mora vsebovati vse lastnosti aplikacije in potrebuje:

- **platformo oziroma operacijski sistem**, ki služi kot vmesnik med strojno opremo naprave in mobilno aplikacijo oziroma igro,
- **sistem za simulacijo igre**, ki skrbi za delovanje in izrisevanje igre,
- **mobilno napravo,** na kateri se igra izvaja in nudi različne načine interakcije z igro.

2.2 Mobilni operacijski sistemi

Programiranje iger za mobilne naprave je bilo včasih bolj omejeno kot sedaj, saj je bila na voljo le J2ME platforma, ki se je izvajala na operacijskih sistemih proizvajalcev telefonov. Z razvojem naprednejše tehnologije so se sčasoma razvile bolj napredne platforme v obliki programske opreme, ki služi kot vmesnik med uporabnikom in strojno opremo naprave. Pod besedo platforma danes razumemo operacijski sistem. Trenutno je na svetu več kot 2,1 milijarde mobilnih naprav s podporo za J2ME platformo, vendar omenjena platforma postaja zastarela, saj so jo zasenčile novejše mobilne platforme [3,4,5].

Ker se mobilni operacijski sistemi med seboj razlikujejo, je potrebno vsako igro prilagoditi posameznemu operacijskemu sistemu. Operacijski sistemi so napisani v različnih programskih jezikih, imajo različne aplikativne modele in vsak se izvaja na svoj lasten način. Vendar je ne glede na uporabljen programski jezik zasnova izdelave igre za vsako platformo oziroma operacijski sistem enaka, le implementacija je drugačna.

Primarni cilj mobilnega operacijskega sistema je opravljanje telefonskih klicev, zato so vse platforme prilagojene temu, da je na mobilniku vedno dovolj pomnilnika za sprejemanje klicev in podobna opravila. S tem namenom so bili razviti različni aplikativni modeli, ki se razlikujejo med posameznimi platformami.

Trenutno se uporabljajo naslednji operacijski sistemi [3] (njihova zastopanost na mobilnih napravah v svetovnem merilu je navedena v oklepajih, podatki za prvo četrtletje leta 2011):

- Android (36 %)
- Symbian (27 %)
- iOS (17 %)
- RIM-blackberry (13 %)
- Windows phone (4 %)
- Ostali (3 %)

2.3 Sistem za simulacijo igre

2.3.1 Komponente simulacije

Vsaka igra potrebuje sistem za simulacijo igre, ki simulira igro na podlagi različnih paramaterov. Pod sistemom razumemo različne komponente, ki sestavljajo igro in so med seboj povezane.

Glavne komponente igre so:

- **Glavni meni** – zaslon, kjer upravljamo parametre igre,
- **Elementi igre** – karakterji, nasprotniki, ozadje, število točk ipd.,
- **Način interakcije z igro** – zaslon, tipkovnica (fizična ali virtualna) ipd.
- **Zanka programske kode (gameloop)** – povezuje elemente igre z interkacijo uporabnika, jih posodablja in izrisuje.

Poleg glavnih so lahko potrebne tudi naslednje komponente:

- **Zbirke elementov**: to so vsi elementi, ki sestavljajo igro, tudi tisti, ki niso neprestano vidni na zaslonu.
- **Sistem za prikazovanje elementov igre**: to je način prikazovanja naših elementov igre. Prikazujemo jih lahko v 2D ali v 3D prostoru in iz različnih zornih kotov uporabnika.
- **Sistem za posodabljanje stanja elementov igre**: ta komponenta skrbi za posodobitev vseh stanj elementov igre. Stanje je na primer pozicija objekta glede na njegovo hitrost ali posodobitev animacije objekta.
- **Sistem za predvajanje zvokov**: to je sistem, ki skrbi za predvajanje in manipulacijo zvokov med igro, pa tudi uporaba vibracij.
- **Orodja za komunikacijo**: ta orodja so del platforme, ki omogoča razpolaganje z vsemi možnimi načini za komunikacijo dveh mobilnih naprav preko »zraka«. Sem uvrščamo komunikacijo preko mobilnega in brezžičnega omrežja ter preko različnih protokolov, kot je na primer bluetooth.
- **Dostop do vseh sistemskih virov na mobilni napravi**: dostop je del platforme, ki vsebuje vse potrebne vmesnike API za manipulacijo sistemskih virov telefona, kot so GPS, video kamera, merilec pospeškov (ang. Accelometer), podatkovne baze in podobno.
- **Sistem za zaznavanje trkov** (ang. collision detection): sistem, ki skrbi za zaznavanje trkov, saj je včasih potrebno zaznati, kdaj se dva objekta dotakneta.

Ni nujno, da vsaka igra vključuje vse komponente, lahko vključuje samo nekatere, vendar brez elementov igre, zanke programske kode (gameloop) in načina interakcije z igro ne gre. Od ideje igre oziroma njenih potreb je odvisno, katere komponente igre bodo uporabljene in kako bodo implementirane. Poleg vseh naštetih komponent je potrebna še komponenta, ki skrbi, da igra deluje tako, kot si je razvijalec igre zamislil. Tej komponenti pravimo **logika igre.**

Za zadovoljstvo končnega uporabnika mobilne igre je potrebno, da je igra preprosta za uporabo, zanimiva, pregledna ter konsistentna. Konsistentno delovanje je pri igrah nujno potrebno, da se igre lahko izvajajo na različnih vrstah strojne opreme. To pomeni, da se v primeru nepravilne implementacije igra odvija hitreje na zmogljivejših napravah, na manj zmogljivih pa počasneje.

Da bi si naštete komponente laže predstavljali, se vrnimo k primeru igre Kača. Igro sestavljajo naslednje komponente:

- **Glavni meni**, kjer so nastavitve za zvok, pregled najboljših rezultatov igre in možnost igranja nove igre ali nadaljevanja igre,
- **Način interakcije z igro**, kjer se zaznava pritisk smerne tipke kot navodilo za premik kače na igralni površini,
- **Zbirka elementov igre**, ki vsebuje kačo in njene člene, zid, ki predstavlja mejo igralne površine ter objekte, ki predstavljajo hrano oziroma cilj igre,
- **Sistem za prikazovanje elementov**, ki prikazuje zbirko elementov igre na zaslonu (v 2D ali 3D prostoru),
- **Sistem za posodabljanje stanj elementov**, ki skrbi, da se kača premika in da je premik kače na različno zmogljivih napravah vedno enak,
- **Sistem za zaznavanje trkov**, ki zaznava, ali se je kača zaletela v zid, sama vase ali v hrano
- **Sistem za predvajanje zvokov**, ki predvaja zvoke ob posameznih ključnih dogodkih v igri.

2.3.2 Mobilni pogoni igre

Ker je postopek razvoja vseh komponent igre časovno potraten, obstajajo različni vmesniki API oziroma pogoni igre (ang. game engine), ki nudijo vse komponente, potrebne za izdelavo igre. Pogoni omogočajo izdelavo vseh zvrsti iger, nekateri pa tudi razvoj iger neodvisno od platforme oziroma operacijskega sistema. To pomeni, da lahko razvijemo eno samo različico igre, ki bo delovala na različnih platformah.

Z uporabo tovrstnih vmesnikov API se razvijalcu ni potrebno ukvarjati z različnimi metodami realizacije igre in njenih komponent ali s prilagajanjem na različno strojno opremo. Za razvoj igre je potrebno že izdelane vmesnike API uporabiti brez poznanja njihovega delovanja. To olajša razvoj, vendar ga pa lahko tudi oteži. V primeru, da ne dosežemo želega delovanja oziroma rezultatov, se je potrebno poglobiti v delovanje pogona, da ugotovimo izvor težav. To pa je lahko težavno, saj je zato potrebno razumeti njihovo delovanje in povezanost.

Na voljo je velika izbira mobilnih pogonov, spodaj pa so našteti najpogostejši:

- **COCOS2d:** open source pogon, ki omogoča razvoj mobilnih iger za iPhone in Android naprave,
- **Corona:** plačljiv pogon, ki omogoča razvoj mobilnih aplikacij za iPhone, iPad in Android naprave,
- **AndEngine:** open source pogon, ki omogoča razvoj mobilnih iger v 2D prostoru za Android naprave in
- **Unity:** plačljiv pogon, ki omogoča razvoj mobilnih iger v 3D prostoru za iPhone naprave.

2.4 Mobilne naprave

Mobilna igra se od ostalih iger razlikuje po napravi, na kateri se izvaja. Poleg čim bolj učinkovitega opravljanja glavnih funkcij naj bi bile mobilne naprave tudi čim manjše in čim bolj mobilne, zato je temu je prilagojena tudi strojna oprema. V primerjavi z namiznim računalnikom je strojna oprema omejena z baterijo in s tem tudi posamezne strojne komponte naprave. Zaradi tega so mobilne naprave počasnejše in energijsko manj potrošne kot namizni računalniki.

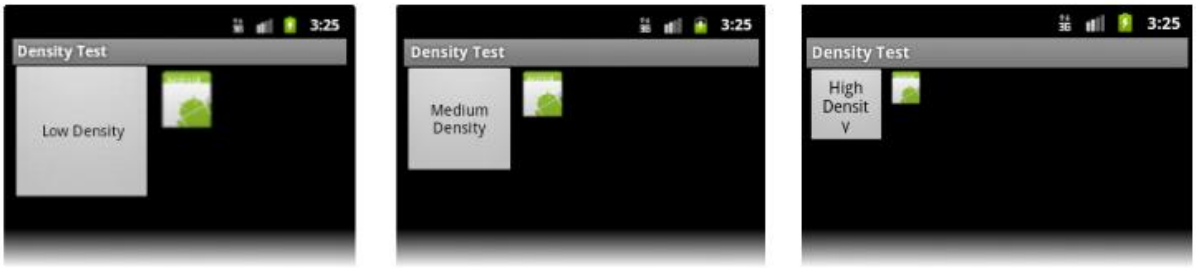
Operacijski sistemi so namenjeni uporabi na raznovrstnih napravah, zato mora biti mobilna igra prilagojena vsaki napravi, platforma pa mora nuditi mehanizme za prilagajanje.

Da dosežemo optimizirano igro, moramo najprej dobro poznati **platformo,** za katero igro razvijamo. To ne pomeni, da jo moramo poznati do potankosti, nujno pa je vedeti, kako aplikacija deluje, kako komunicira s sistemom in kako doseči želene rezultate. Vedno je koristno podrobno raziskati platformo še pred izdelavo same igre, ker se sicer sredi razvoja lahko zgodi, da moramo zaradi slabega začetnega poznavanja platforme prestrukturirati svojo programsko kodo.

Pri izdelavi igre moramo upoštevati različne značilnosti mobilnih naprav, kot so na primer zaslon, različno sposobna strojna oprema in možni načini interakcije z igro. Te značilnosti pa se razlikujeje od naprave do naprave

Zasloni mobilnih naprav se ločijo po fizičnih lastnostih, med katere štejemo velikost zaslona (velikost diagonale v inčih), ločljivost zaslona (število točk na zaslonu - px) in gostoto točk (DPI – dots per inch). Pri zaslonih je najpomembnejša lastnost DPI, kar pomeni, da imamo lahko dva različna zaslona enake velikosti in z enako ločljivostjo, vendar z različno gostoto točk. Tako na primer ob izrisu 100 točk dolgega elementa

opazimo, da je prikaz elementa na zaslonu z večjo gostoto točk bistveno manjši kot na zaslonu z manjšo gostoto točk (slika 1) [10,11].



Slika 1: Pogled na tri zaslone z različnimi gostotami točk - 120 DPI, 160 DPI in 240 DPI.

V današnjem času imajo nekatere naprave vgrajen **grafični procesor**(GPU)**,** ki omogoča, da se izrisovanje in preračunavanje izvaja na grafičnemu procesorju in ne na glavnem procesorju. To dejansko privede do pohitritve celotnega procesa in obenem omogoča izdelavo grafično bolj zahtevnih iger, katerih izdelava zaradi slabše strojne opreme prej ni bila mogoča.

Ker je mobilna naprava omejena z velikostjo **pomnilniškega prostora**, se lahko zgodi, da ob nepravilni implementaciji igre napravi zmanjka pomnilniškega prostora, kar prisili platformo oziroma operacijski sistem, da pomnilniški prostor sprosti, kjer ni več potreben (ang. garbage collection). Ker je to opravilo dolgotrajno (~100-300ms), lahko vpliva na odzivnost igre in s tem povzroči upad števila izvedb zanke igre. Pri hitrejših igrah je to še posebej pomembno, saj v primerjavi s počasnimi igrami pomnilniškega prostorja hitreje zmanjka.

To še posebej velja za elemente, ki uporabljajo slike za predstavitev elementa na zaslonu. Da sistem sliko prikaže, jo mora najprej poiskati, jo prebrati in jo naložiti v pomnilnik. To je časovno zahtevno opravilo, ki blokira nadaljnje izvajanje, kar lahko pripelje do padca števila izvedb zanke.

Da se izognemo tovrstnim težavam, vsem objektom, ki so uporabljeni v igri, dodelimo (ang. memory allocation) pomnilnik pred dejanskim pričetkom igre, tudi če se večino časa med igro ne uporabljajo. S tem nam med igro ni potrebno dodeljevati novega prostora v pomnilniku, kar pomeni, da sistem ne bo prisiljen med igro sproščati pomnilniškega prostora. Pomnilniški prostor po potrebi sprostimo, ko igra preneha z izvajanjem.

Testiranje igre predstavlja pomembno fazo razvoja igre. Igre testiramo na emulatorju platforme ali na fizični napravi. Najbolj priporočljivo je izvesti testiranje na večjem

številu različnih fizičnih naprav, saj emulator teče na operacijskem sistemu računalnika, zato s pomočjo emulatorja dobimo drugačne rezultate hitrosti in odzivnosti, kot bi jih dobili, če bi opravljali testiranje na dejanskih napravah, za katere igro razvijamo. Le tako lahko preizkusimo zanesljivost igre in preverimo, kako se prilagaja različnim napravam.

Ker se mobilne naprave razlikujejo po vrsti **strojne opreme**, so nekatere zato hitrejše od drugih, vendar pa mora igra delovati konsistentno na vseh mobilnih napravah. Cilj uspešne implementacije igre je najti ravnovesje med posodabljanjem stanja in izrisovanjem stanja igre, da dosežemo enako delovanje na različno zmogljivih napravah. V nasprotnem primeru pride do nekonsistentnosti.

Če za primer ponovno vzamemo igro Kača in premik objekta, se pravi kače, lahko vidimo, da pride do odstopanja med potekom igre na hitrejših in počasnejših napravah. Recimo, da lahko visoko zmogljiva naprava izvede posodobitev in izris stanja oziroma. zanko 30-krat na sekundo, slabše zmogljiva naprava pa 15-krat na sekundo. Če v vsaki iteraciji zanke premaknemo kačo na zaslonu za 1 enoto, se bo na slabše zmogljivi napravi kača premaknila za 15 enot, na hitrejši pa za 30 enot.

2.5 Delovanje igre

2.5.1 Preprosta zanka

Da laže razumemo delovanje igre, jo lahko primerjamo s preprostim filmskim posnetkom. Posnetek prikaže 24 sličic na sekundo (FPS), kar pomeni, da se v eni sekundi slika zamenja 24-krat. Posnetek izgleda tekoč in ne opazimo, da se slike dejansko menjavajo z isto časovno periodo. Enak pristop velja pri igrah. Da dosežemo premik objekta po zaslonu in ga zraven še animiramo, moramo zaslon in stanje igre osvežiti vsaj 24-krat, da uporabnik ne opazi menjave animacije ali negladkega premika karakterja.

Kot smo že omenili, mobilna igra deluje tako, da izvaja isto programsko kodo, imenovano zanka ali »gameloop«. Zanka je kot utrip srca vsake igre, nobena igra ne more delovati brez nje. Vsaka igra vsebuje sekvenco ključnih dogodkov:

1. zaznavanje interakcije uporabnika z igro,
2. posodabljanje stanja igre,
3. izvedba umetne inteligence,
4. predvajanje glasbe in zvočnih efektov,
5. osveževanje (izrisovanje) stanja igre.

Sekvenco ključnih dogodkov upravlja zanka programske kode. Če vsa opravila poenostavimo, jih lahko razdelimo na dve metodi: **posodabljanje stanja** igre in **prikazovanje stanja** igre. V najpreprostejši obliki, zapisani v pseudo kodi, zanka izgleda tako:

```
while(running) {
        updateGameState();
        displayGameState();
}
```

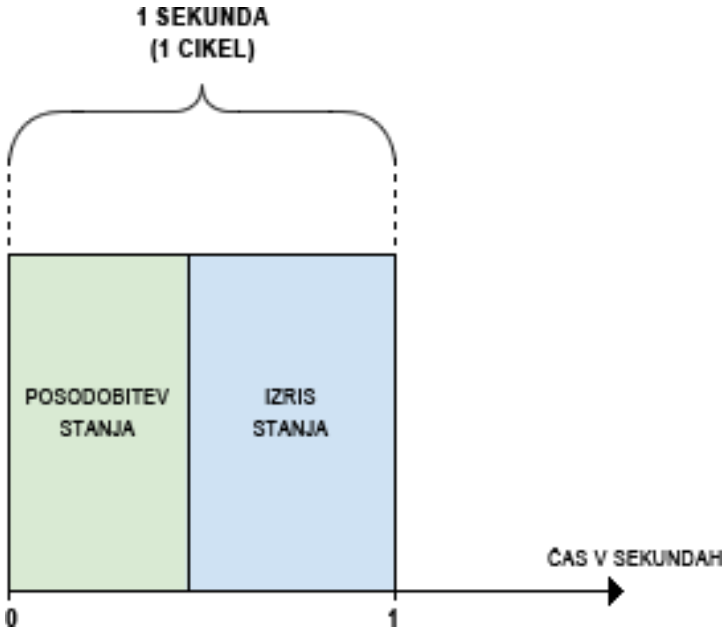
Funkcija updateGameState v zgornjem zapisu posodobi stanje vseh elementov igre, kot je na primer preračun pozicije elementa na podlagi hitrosti ali drugih fizikalnih lastnosti. Funkcija drawGameState pa izriše vse elemente igre na zaslonu. V kakšnem prostoru (2D ali 3D) in iz kakšnega zornega kota uporabnika se bo igra prikazovala, pa je odvisno od same igre.

Problem pri taki implementaciji je ta, da ne upošteva niti časa niti sredstev, igra se preprosto izvaja. To pomeni, da se zanka izvede na hitrejših napravah večkrat na časovno enoto kot na počasnejših.

Kako igra deluje, nam pomagajo razumeti naslednji trije pojmi:

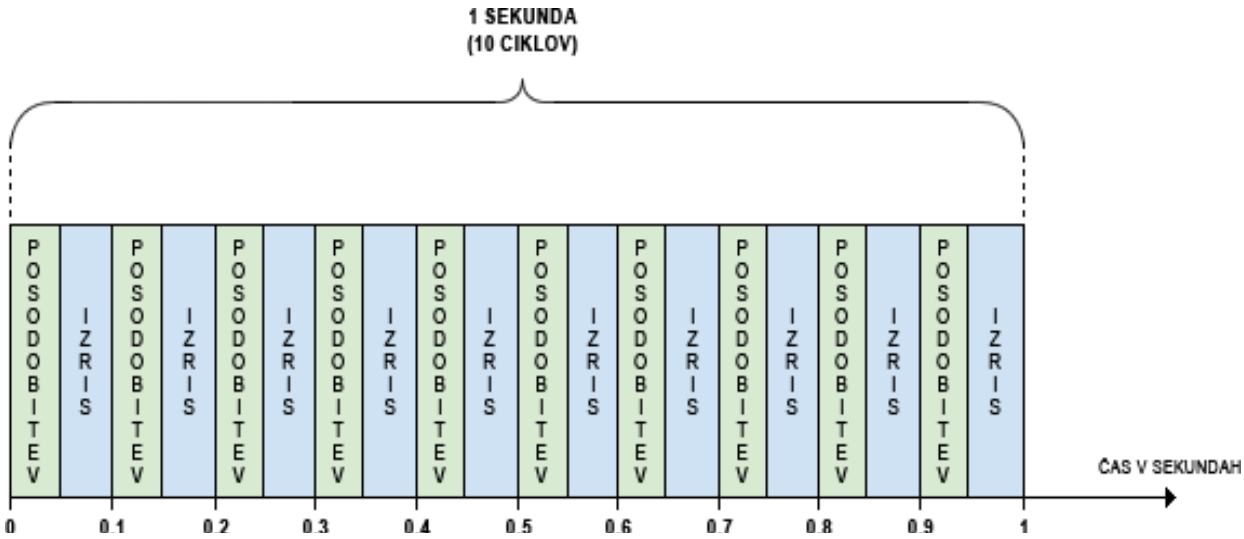
- **sličice na sekundo** (ang. Frames per second-FPS): označuje število izrisov elementov igre na sekundo. V povezavi z zgornjim zapisom zanke to pomeni število klicev metode drawGameState na sekundo.
- **posodobitev na sekundo** (ang. Updates per second-UPS): označuje število posodobitev stanja na sekundo oziroma ***hitrost igre***. V povezavi z zgornjim zapisom zanke to pomeni število klicev metode updateGameState na sekundo.
- **cikel ali okvir** (ang. Frame): označuje izvedbo ene iteracije zanke.

Spodnja slika (slika 2) prikazuje 1 FPS, kjer cikel posodobitve-izrisa traja natanko 1 sekundo, kar pomeni, da se slika na zaslonu spremeni oziroma ponovno izriše vsako sekundo.



Slika 2: Prikazan 1 FPS.

Na spodnji sliki (slika 3) se prikazuje 10 FPS, cikel posodobitev-izrisa traja 100 ms, kar pomeni, da se slika na zaslonu ponovno izriše vsako desetinko sekunde.



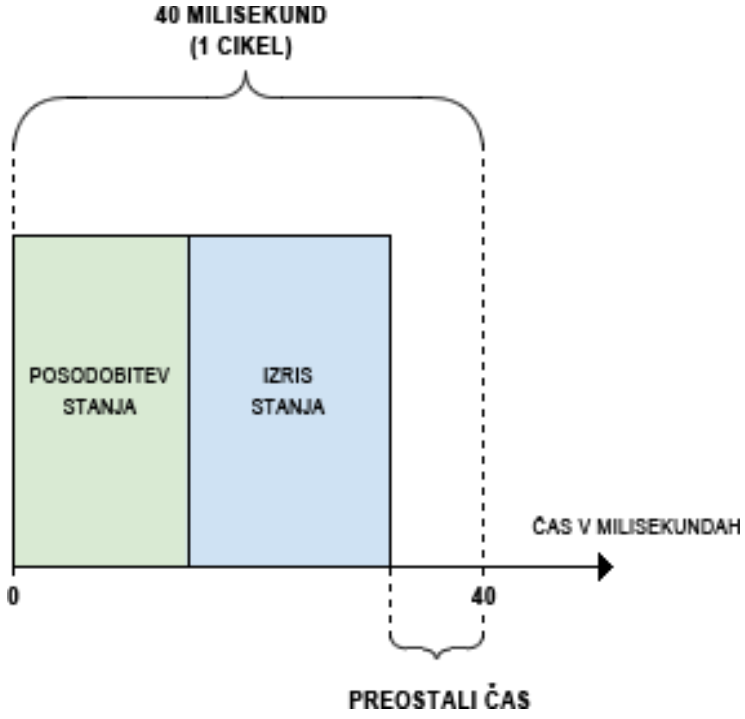
Slika 3: Prikazanih 10 FPS

Če se cikel posodobitve – izrisa vedno izvede točno v eni desetinki sekunde, lahko to predstavlja težavo predvsem pri prvoosebnih strelskih igrah ali igrah, kjer je eden od možnih scenarijev tudi ta, da se na igralni površini znajde večje število »sovražnikov«, ki streljajo v nas. V eni sami posodobitvi je potrebno za vsakega »sovražnika« in vse izstrelke posebej posodobiti stanje oziroma pozicijo, potrebno pa je tudi preveriti dotike (ang. Collision detection). Situacija se spremeni, če sta na

zaslonu le 2 nasprotnika. Zaradi tega ne moremo predpostaviti, da se bo skozi celotno igro cikel izvedel v 1/10 sekunde.

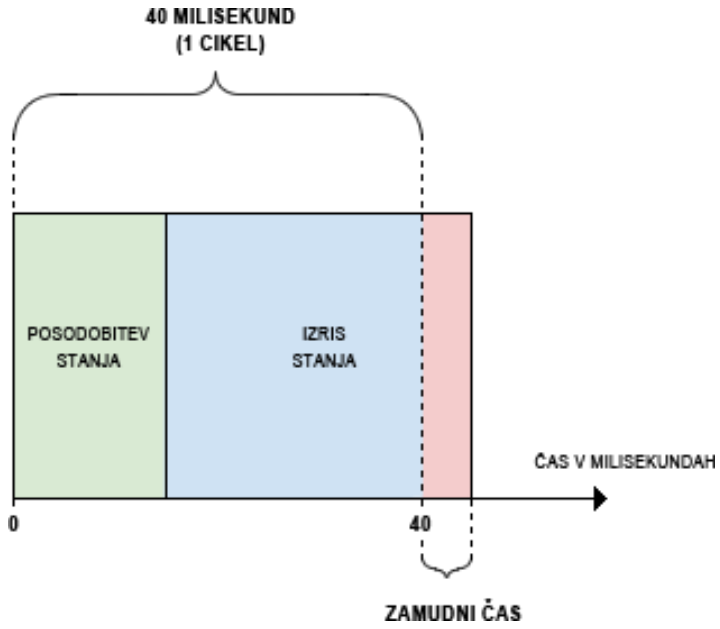
Če želimo, da igra deluje tako kot filmski posnetek s 25 FPS, moramo poklicati metodo displayGameState() vsakih 40 ms (1 s = 1000 ms; 1000/25 = 40 ms). Pri tem moramo upoštevati, da se pred metodo kliče updateGameState(), in če želimo doseči 25FPS, moramo celotni cikel posodobitve – izrisa izvesti v točno 40 ms. V nasprotnem primeru dosežemo več oziroma manj FPS in UPS.

Spodnja slika (slika 4) prikazuje cikel, ki se konča v manj kot 40 ms, kar pomeni, da ostane nekaj časa pred izvedbo naslednjega cikla in se bo zaradi tega zanka izvedla večkrat, kot smo predpostavili. S tem bo igra delovala hitreje kot 25 FPS.



Slika 4: Predčasno zaključen cikel.

Spodnja slika (slika 5) prikazuje cikel, ki se zaključi po več kot 40 ms. To pomeni, da je čas, ki je potreben za izvedbo cikla večji kot želeni. Če cikel za izvedbo potrebuje na primer 42 ms, pomeni, da zaostaja za 2 ms in s tem ne dosežemo želenih 25 FPS.



Slika 5: Cikel, ki se ne zaključi pravočasno.

2.5.2 Razširitve zanke

Kot je zgoraj prikazano, čas izvedbe enega cikla nikoli ni enak, še posebej pa ne na različno zmogljivih in obremenjenih napravah. Zato je potrebno začetno zanko prilagoditi. Spodaj navedene metode opisujejo različne načine prilagoditve zanke in vpliv, ki jih imajo prilagoditve na počasne in hitre mobilne naprave [6, 7, 8, 9].

2.5.2.1 Število FPS odvisno od konstantne hitrosti igre

Preprosta rešitev zgornjega problema bi bila, da bi se zanka izvedla 25-krat na sekundo. Potem bi zanka izgledala takole:

```
final int FRAMES_PER_SECOND = 25;
final int SKIP_TIME = 1000 / FRAMES_PER_SECOND; // 40ms

long next_time = System.currentTimeMillis();

long sleep_time = 0;

while(running) {
        updateGameState();
        displayGameState();

        next_time += SKIP_TIME;
        sleep_time = next_time-System.currentTimeMillis();
        if(sleep_time > 0){
                Thread.sleep(sleep_time);
        } else {
                // zaostajamo
        }
}
```

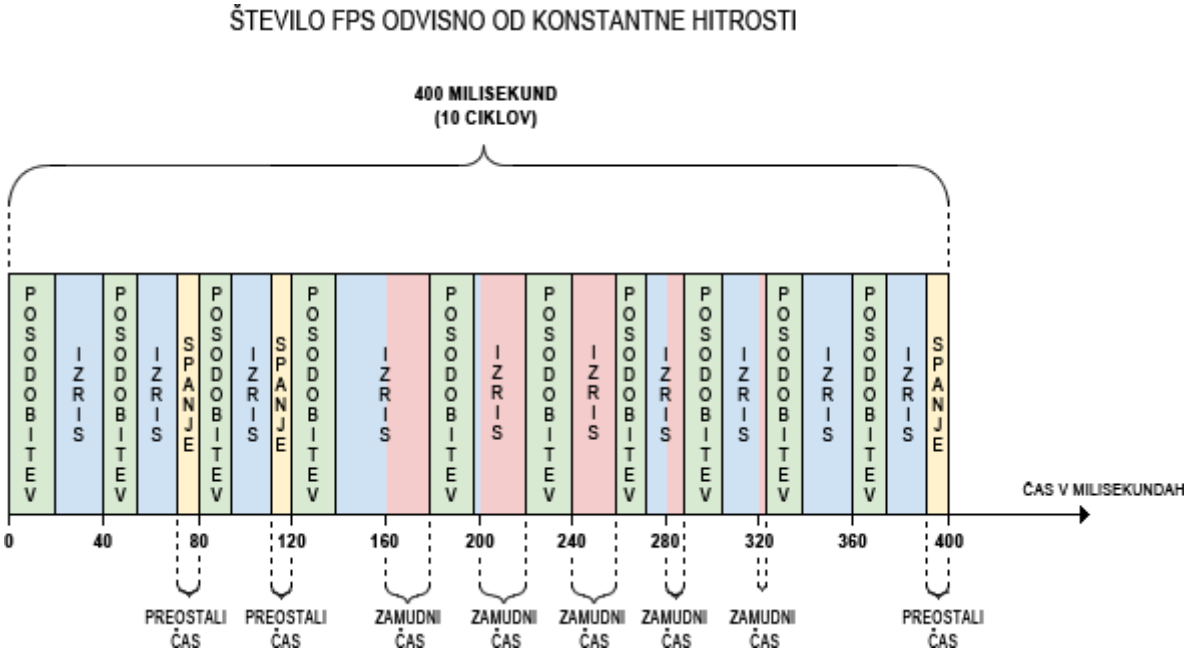
S tako zanko bi se izvedba cikla vedno zgodila 25-krat na sekundo oziroma vsakih 40 ms. Ko cikel potrebuje za izvedbo manj časa, kot ga ima na voljo, se uporabi t.i. spanje (ang. sleep), ki omogoča, da se izvajanje za preostali čas začasno preneha. To pripomore h konsistentnemu delovanju igre in zmanjša porabo baterije naprave. Med testiranjem na fizičnih napravah lahko prilagodimo parameter število sličic na sekundo (FRAMES_PER_SECOND), da dosežemo čim boljšo konsistentnost na različno zmogljivih napravah.

Vendar metoda vseeno ni optimalna in lahko pride do težav, tako na počasnih kot na hitrih napravah. Težava nastane, ko cikel traja dlje kot 40 ms. To se pozna na počasnih napravah, katerih strojna oprema ne dovoljuje oziroma ne zmore prikazati 25 sličic na sekundo. V najslabšem primeru »težji« deli igre delujejo počasneje, ostali deli pa delujejo tekoče. To lahko privede k slabši igralnosti igre, v skrajnih primerih pa tudi do neigralnosti.

Hitre naprave s pomočjo boljše strojne opreme laže prikazujejo večje število sličic na sekundo. Tu pa pride do vprašanja, zakaj bi se zadovoljili s prikazom 25 sličic na sekundo, če lahko naprava brez večjih težav prikaže več kot 100 sličic na sekundo. Zato lahko pri tej metodi pride do slabše izkoriščenosti in manj kakovostnega prikaza igre.

Ta metoda se hitro implementira, kar pomeni, da je tudi programska koda igre dokaj preprosta. Če nastavimo visoko število sličic na sekundo, lahko to povzroči težave na počasnejših napravah, v nasprotnem primeru pa se lahko zgodi, da bomo poslabšali kakovost prikaza oziroma vizualno predstavo igre na hitrejših napravah.

Slika 6 prikazuje delovanje te metode.



Slika 6: Delovanje metode "Število FPS odvisno od konstantne hitrosti igre".

2.5.2.2 Hitrost igre odvisna od variabilnih FPS

Druga metoda zanke je, da se izvaja kar se da hitro in število sličic na sekundo narekuje hitrost igre. Stanje igre se posodablja s časovno razliko med dvema posodobitvama.
Zanka v primeru take implementacije izgleda tako:

```
long prev_time;
long curr_time = System.currentTimeMillis();

while(running) {
        prev_time = curr_time;
        curr_time = System.currentTimeMillis();

        updateGameState(curr_time-prev_time);
        displayGameState();
}
```

S tako zanko je število ciklov na sekundo vedno drugačno, odvisno od sposobnosti in obremenjenosti naprave. Posodobitev stanja postane bolj zapletena, ker je potrebno upoštevati časovno razliko med dvema posodobitvama. Sprva ta metoda deluje kot dobra rešitev, vendar ima težave tako na počasnih kot tudi na hitrih napravah.

Na *počasnih napravah* lahko pride do zamikov med dvema posodobitvama stanja, ko igra postane »težka«. To je še posebej značilno za igre, predstavljene v 3D prostoru.

To poveča časovni razmik med dvema cikloma, ki vpliva na odzivni čas zaznavanja vhodnih enot (ang. input) in s tem tudi podaljša odzivni čas uporabnika. Zaradi tega lahko preprost manever, ko se mora uporabnik izogniti neki oviri na igralni površini, postane nemogoč ali močno otežen. V najhujšem primeru bi oviro kar preskočili (ang. tunneling), saj bi bil časovni razmik tako velik, da bi bila razdalja prepotovanega objekta daljša kot ovira.

Na *hitrih napravah* se pri tej metodi pojavi drugačna težava. Da bi razumeli problem, moramo razumeti, kako deluje predstavitev decimalnih števil v računalniku. Pomnilnik je omejen, zato nekatera števila ne morejo biti predstavljena. Na primer 0.1 se ne more predstaviti binarno in se zato zaokroži, preden se shrani. Vzemimo primer simulacije objekta, recimo avta, ki ima hitrost 1 enoto na sekundo (0.001 enote na ms). Po 10 sekundah je ta avto prevozil razdaljo 10 enot. Če želimo implementirati metodo za izračun razdalje pri različnih hitrostih delovanja zanke, to naredimo s spodnjo metodo, ki kot argument sprejme FPS:

```
public double getDistance(long fps) {
        double skip_ticks = 1000 / fps;
        int total_ticks = 0;
        double distance = 0;
        double speed_per_tick = 0.001;

        while(total_ticks < 10000) {
                distance += speed_per_tick*skip_ticks;
                total_ticks += skip_ticks;
        }
        return distance;
}
```

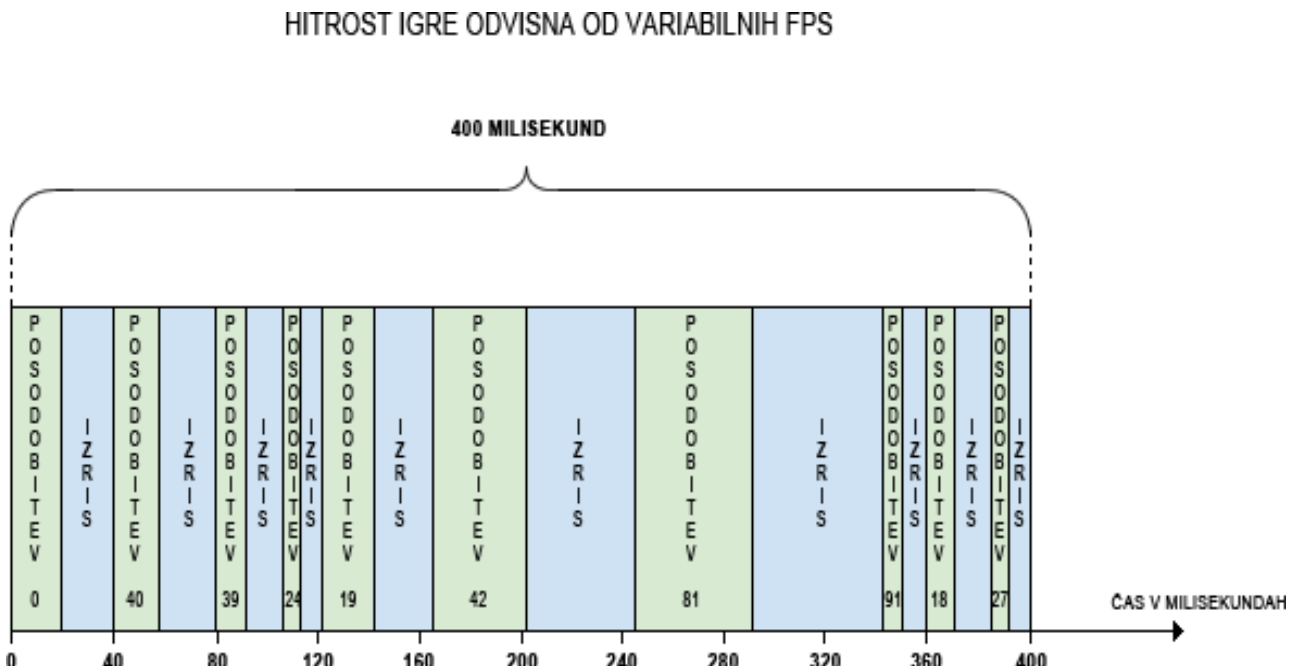
Če metodo testiramo pri različnih FPS, bomo opazili napako:

```
> getDistance(40)
  10.000000000000075

> getDistance(100)
  9.9999999999998312
```

Rezultat testa bi moral biti vedno 10.0, vendar zaradi zaokroževanja dobimo napako. Kljub temu da je napaka zelo majhna, lahko privede do večjih odstopanj pri nadaljnjem računanju. Zato ima ta metoda slabosti pri izvajanju tudi na zmogljivejših napravah.

Slika 7 prikazuje delovanje te metode.



Slika 7: Delovanje metode "Hitrost igre odvisna od variabilnih FPS".

2.5.2.3 Konstantna hitrost igre z maksimalnim FPS

Konstantna hitrost igre z maksimalnim številom FPS je izpopolnjena različica prve opisane metode, kjer je število FPS odvisno od konstantne hitrosti igre. Pri slednji se pojavi problem pri izvajanju igre na slabših napravah, ko naprava ne more doseči želenih FPS. Možna rešitev te težave je, da v primeru zamude cikla stanje ponovno posodobimo in povečamo število posodobitev na sekundo na 50.

Zanka v tem primeru izgleda tako:

```
final int UPDATES_PER_SECOND = 50;
final int SKIP_TICKS = 1000 / UPDATES_PER_SECOND;
final int MAX_FRAMESKIP = 10;

long next_game_tick = System.currentTimeMillis();
int loops = 0;

while(running) {
   loops = 0;
   while( System.currentTimeMillis() > next_game_tick && loops < MAX_FRAMESKIP) {
      updateGameState();
      next_game_tick += SKIP_TICKS;
      loops++;
   }
   displayGameState();
}
```

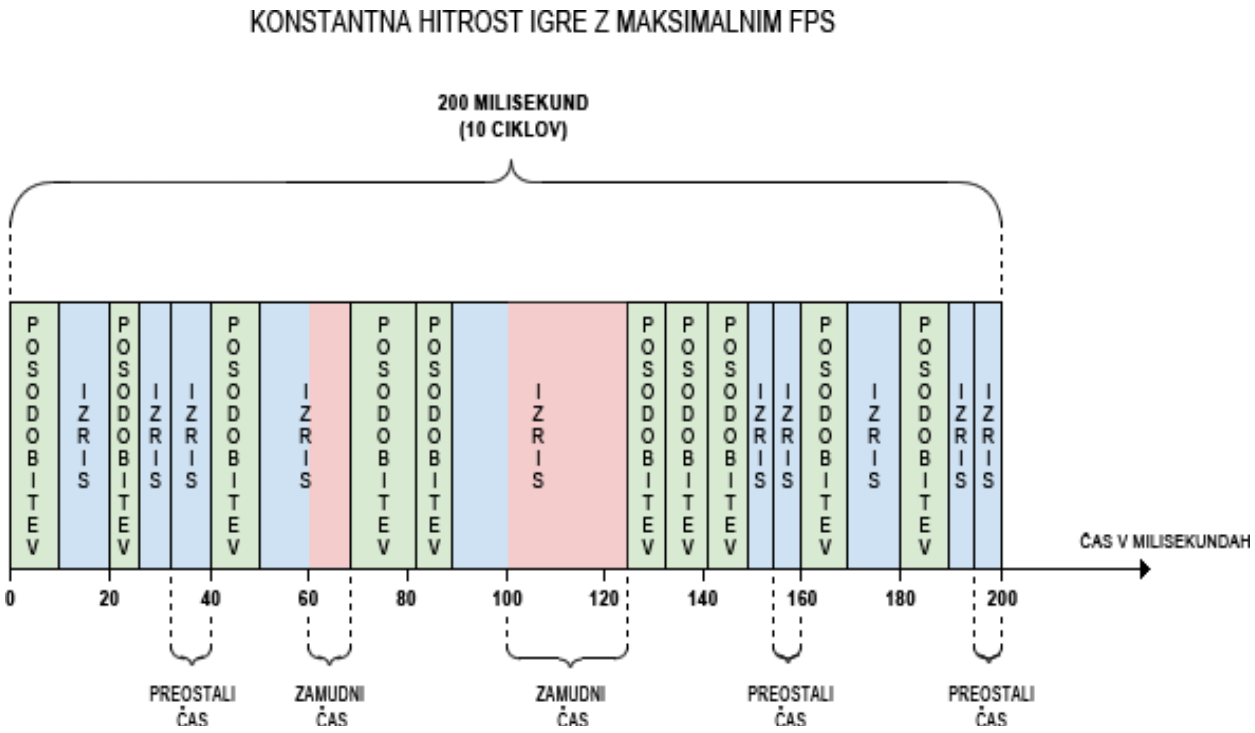
S tako izvedbo zanke bo stanje igre posobljeno do 50-krat na sekundo (odvisno od zmogljivosti naprave), izrisovanje pa bo pa potekalo kar se da hitro glede na preostali čas za izvedbo cikla. Če bo cikel zaostajal, se bo stanje igre ponovno posodobilo, vendar le tolikokrat, kot je definirano v MAX_FRAMESKIP. Če pa bo predčasno končal, bo pa preostali čas izkoristil za ponovni izris.

Na *počasnih napravah* bo FPS padel, vendar igra se bo odvijala do 50 UPS, v kolikor bo to naprava zmogla. V nasprotnem primeru igra ne bo dosegla konsistentnosti, saj ne bo na vseh napravah enako število posodobitev.

Na *hitrejših napravah* igra ne bo imela težav, vendar tako kot pri prvi metodi ni smiselno omejevati zmogljivosti naprave z zgornjo mejo.

Metoda je preprosta za implementacijo in ohranja programsko kodo igre preprosto, vendar, kot smo omenili že pri prvi metodi, lahko pri definiranju visokih FPS še vedno predstavlja težave na počasnih napravah, medtem ko definiranje nizkih FPS povzroči slabšo vizualno predstavo na hitrih napravah.

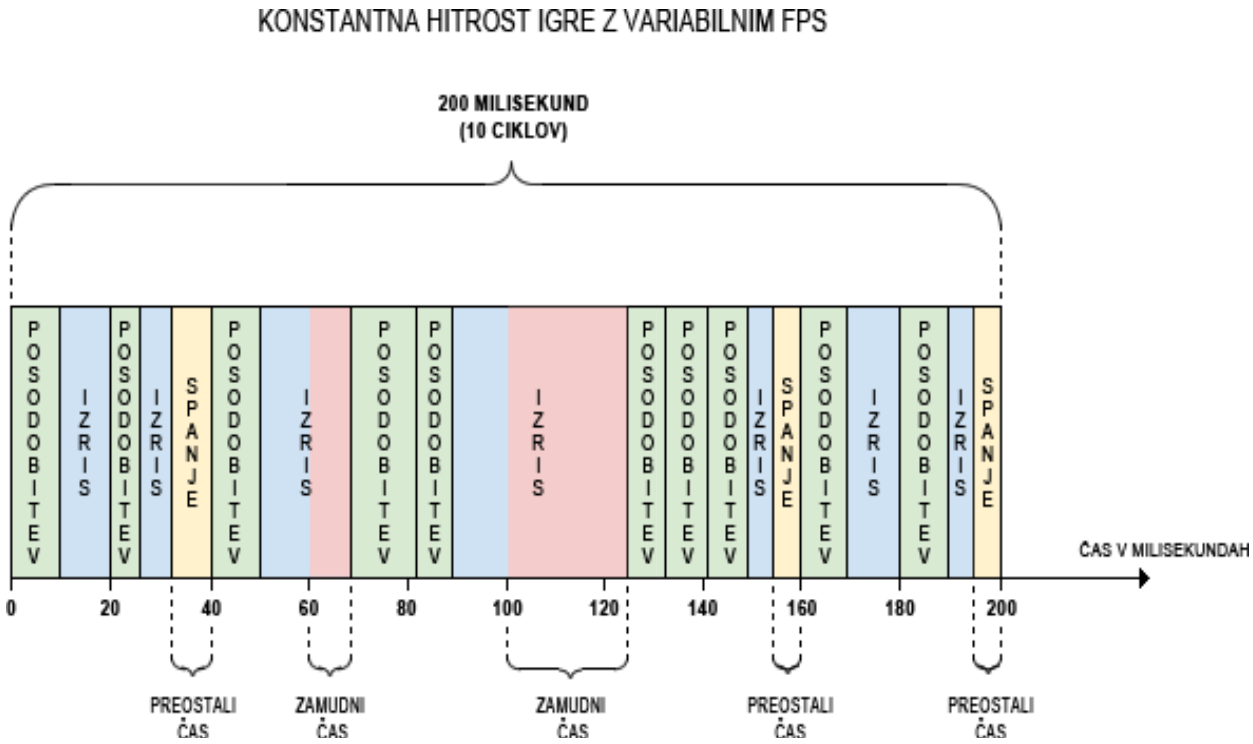
Slika 8 prikazuje delovanje te metode.



Slika 8: Delovanje metode "Konstantna hitrost igre z maksimalnim FPS".

Konstanta hitrost igre z variabilnim FPS

Ta metoda je skoraj identična kot predhodna, razlika je le v tem, da v primeru predčasnega končanja cikla uporabimo spanje iz prve metode, da pridobimo konstantno hitrost igre. Na spodnji sliki (slika 9) je ta metoda prikazana.



Slika 9: Delovanje metode »Konstantna hitrost igre z variabilnim FPS«.

2.5.2.4 Konstantna hitrost igre neodvisna od FPS

Da bi metodo Konstanta hitrost igre z maximalnim FPS, dodatno izpopolnili, moramo doseči hitrejše delovanje na počasnih napravah in izkoristiti celotno zmogljivost hitrih naprav .
Da bi to dosegli, potrebujemo namesto 50 posodobitev stanj na sekundo le 25 posodobitev na sekundo. Vse zaznavanje interakcije z igro, umetna inteligenca (AI), posodabljanje pozicij in hitrosti elementov potrebuje samo 25 posodobitev, izrisovanje pa se mora izvesti tolikokrat, kot je posamezna naprava sposobna.

```
final int TICKS_PER_SECOND = 25;
final int SKIP_TICKS = 1000 / TICKS_PER_SECOND;
final int MAX_FRAMESKIP = 5;

long next_game_tick = System.currentTimeMillis();
int loops = 0;
float interpolation;

while(running) {
        loops = 0;

        while( System.currentTimeMillis() > next_game_tick && loops < MAX_FRAMESKIP) {
            updateGameState();
            next_game_tick += SKIP_TICKS;
            loops++;
        }
        interpolation = (float)(System.currentTimeMillis() + SKIP_TICKS - next_game_tick )/
(float)(SKIP_TICKS);
        displayGameState(interpolation);
  }
```

S tako implementacijo zanke metoda updateGameState () ostane preprosta in se izvede 25-krat na sekundo oziroma vsakih 40 ms, metoda displayGameState () pa postane bolj zapletena. Izrisovanje je potrebno preurediti tako, da sprejme interpolacijo kot argument. Interpolacija je decimalno število med 0 in 1, ki pove, kje točno med dvema posodobitvama se izrisovanje izvaja.

Če vzamemo za primer objekt, ki se pri vsaki posodobitvi stanja premakne za 10 enot, bo pozicija objekta v 10. ciklu 100, v 11. 110 itd. Ko se slika izrisuje, je na mestu med dvema posodobitvama oziroma cikloma. V času izrisovanja lahko uporabimo pozicijo iz cikla 10, ki je 100, ali pa s pomočjo interpolacije izračunamo točno lokacijo objekta. To pomeni, če smo med ciklom 10 in 11 z interpolacijo 0.3, bi se objekt izrisal namesto na poziciji 10 na poziciji 13 (10+10*0.3 = 13). To bi pripomoglo k boljši vizualni predstavi premikajočega se elementa.

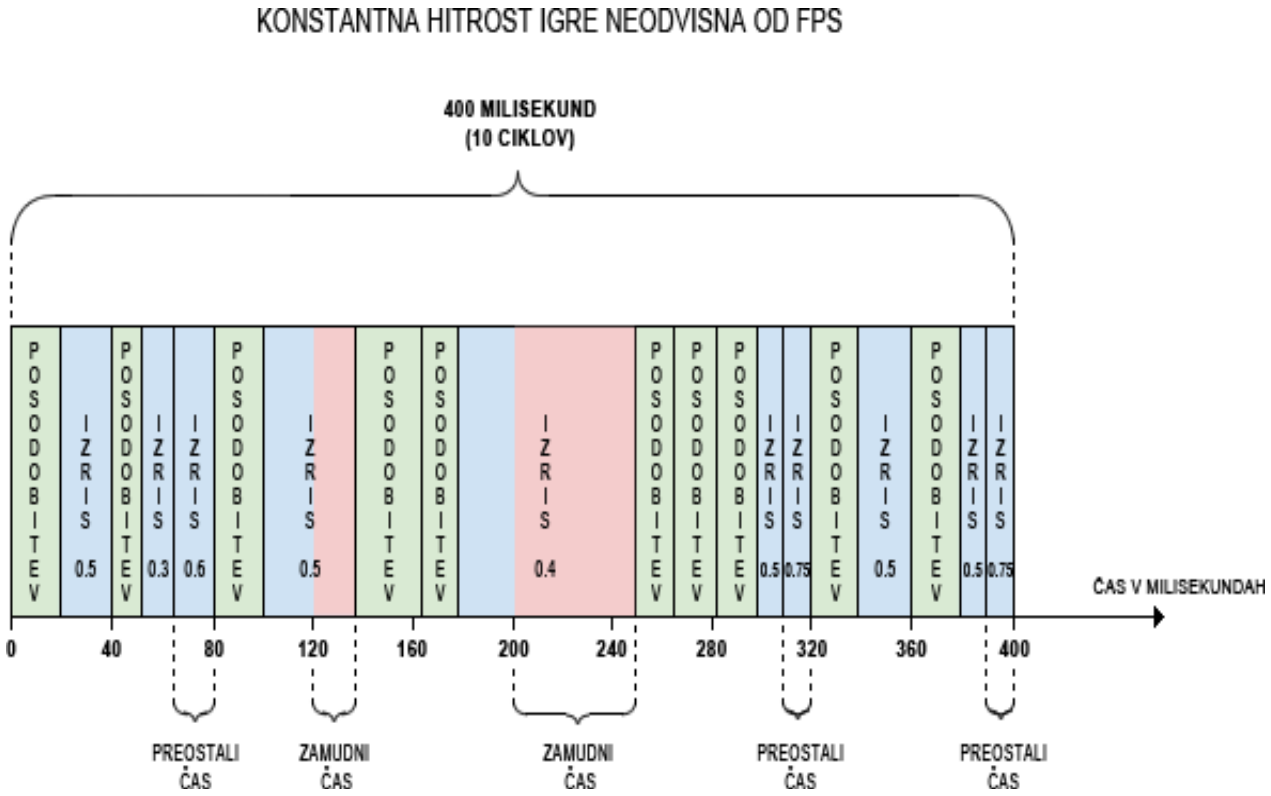
Če pri izrisevanju interpolacije ne uporabimo, se igra odvija s 25 FPS, v nasprotnem primeru pa bo uporabniku dala občutek, kot da se odvija z veliko večjim številom FPS. Uporaba interpolacije je smiselna le takrat, ko so na zaslonu objekti, ki se hitro premikajo, pri statičnih elementih pa interpolacija nima vpliva.

Na *počasnih napravah* lahko v večini primerov predpostavimo, da posodobitev stanja traja manj časa kot izris stanja. S tem lahko tudi predpostavimo, da je igra sposobna delovati na 25 UPS, pa čeprav se zaslon osveži le 15-krat na sekundo.

Na hitrih napravah igra deluje s konstantno hitrostjo 25 UPS, izrisovanje pa je veliko hitreje kot UPS. Zaradi interpolacije bo videti, kot da igra deluje z visokim UPS.

Od vseh opisanih metod je ta metoda najbolj učinkovita in omogoča najboljši izkoristek strojne opreme posamezne naprave. Čeprav je lahko metoda konstantih FPS hitra in preprosta rešitev za mobilne naprave, je ta metoda boljša rešitev, če ima igra hitro premikajoče se elemente in želi napravo maksimalno izkoristiti.

Slika 10 prikazuje delovanje te metode.



Slika 10: Delovanje metode "Konstantna hitrost igre neodvisna od FPS".

3 Android

3.1 Razvoj aplikacij v Androidu

Za uspešen razvoj mobilne igre za operacijski sistem Android je potrebno razumeti delovanje aplikacije znotraj operacijskega sistema oziroma njen življenjski cikel ter z njim povezanih dogodkov.

Za razvoj aplikacij je potrebno razvijalsko okolje oziroma tako imenovani IDE (ang. integrated development enviroment), kamor se piše programska koda. Da bi programska koda imela učinek, pa je potrebna logika, ki je odgovorna za prevod programske kode v format, ki je razumljiv platformi. Za razvoj aplikacij za platformo Android moramo poleg razvojnega okolja imeti tudi Android SDK (ang. Software Developers Kit) oziroma pripomočke za razvijalce programske opreme. Ti pripomočki vsebujejo vse knjižnice, ki so potrebne za nemoteno izvajanje aplikacij na napravah, ki uporabljajo platformo Android. Vsaka programska oprema, ki jo razvijemo za določeno platformo, mora biti preizkušena, preden smo kot razvijalci z njo tudi zadovoljni. V ta namen imamo pri platformi Android na voljo emulator, s pomočjo katerega lahko preizkušamo razvito programsko opremo na namiznem računalniku ter na ta način tudi ugotovimo morebitne napake oziroma odstopanja v kodi. Eno od pomembnejših orodij, ki ga uporabljamo v razvojni fazi, pa je vsekakor orodje za razhroščevanje kode. S pomočjo tega orodja dobimo vpogled v samo delovanje naše aplikacije in tudi odpravimo morebitne napake v kodi.

3.2 Splošno o Androidu

Android je sklad programske opreme za mobilne naprave, ki vključujujejo operacijski sistem, vmesno opremo (ang. middleware) in aplikacije, ki so potrebne za nemoteno delovanje mobilne naprave. Razvit je bil leta 2003 v podjetju Android, ki ga je leta 2005 kupilo podjetje Google. Prva uradna različica je bila izdana leta 2008, in sicer kot odprtokodni operacijski sistem [10,11].

Prednost platforme Android je v ogromni svetovni skupnosti programerjev, ki razvijajo aplikacije, s katerimi se razširja in bogati funkcionalnost mobilnih naprav. Trenutno je za Android na voljo več kot 200.000 aplikacij, ki so na razpolago prek spletne storitve »Android market«. To je spletna trgovina oziroma knjižnica, ki jo upravlja Google. Poleg omenjenega »Android market-a« pa so aplikacije na voljo tudi prek tako imenovanih tretjih ponudnikov. Sam »Android market« pa je v bistvu aplikacija, ki je

prednaložena na večino naprav z operacijskim sistemom Android in uporabnikom omogoča brskanje in nalaganje aplikacij različnih razvijalcev.

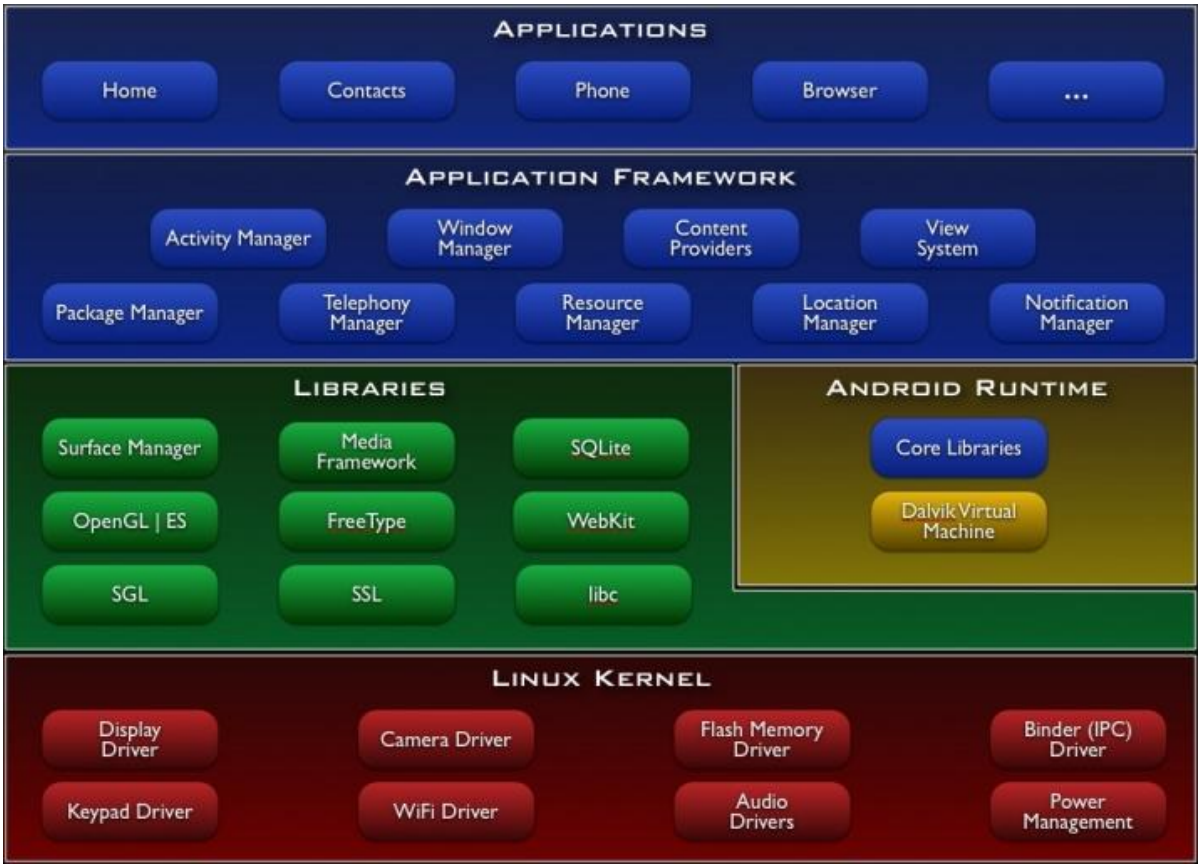
Aplikacije za Android se običajno piše v programskem jeziku Java, medtem ko imajo novejše različice Androida podporo za razvoj v programskem jeziku C ali C++.

Glavne značilnosti Androida so:

- Aplikativno ogrodje – omogoča ponovno uporabo in zamenjavo komponent,
- Dalvik virtualna naprava – optimizirana za mobilne naprave,
- Integriran brskalnik, ki temelji na odprtokodnem webKit pogonu (ang. web engine),
- Optimizirana grafika – poganjajo jo prilagojene 2D grafične knjižnice in 3D grafike, katerih osnovo predstavlja openGL ES 1.0 specifikacija,
- SQLite za shranjevanje podatkov,
- Podpora za multimedijske vsebine – audio/video/slika (WebM, H.263, MPEG-4, SP, H.264, MP3, AAC,HE-AC, AMR, AMR-WB, MIDI, OGG, FLAC, WAV, JPG, PNG, GIF, BMP),
- Povezljivost – GSM/EDGE, IDEN, CDMA, EV-DO, UMTS, bluetooth, brezžična povezava, LTE, NFC in WiMAX,
- Podpora za strojno opremo – video kamera, GPS, zaslon na dotik, kompas, merilec pospeškov, žiroskop, termometer itd.,
- Podpora za več dotikov hkrati,
- Večopravilnost,
- Uporabljanje naprave z govorom – od verzije 2.2 dalje,
- »Tethering« – omogoča napravi, da se uporablja kot dostopna točka za žično in brezžično povezavo, in
- Bogato razvijalsko okolje, ki vključuje emulator naprave, orodja za razhroščevanje, profiliranje pomnilnika in zmogljivosti in vtič za razvijanje v programskem okolju Eclipse IDE.

3.2.1 Arhitektura Androida

Slika 11 [11] predstavlja sklad programske opreme operacijskega sistema Android. Vsak sloj je v nadaljevanju razložen od najnižjega nivoja do najvišjega.



Slika 11: Arhitektura Androida.

Jedro Linux

Android deluje na različici jedro Linux 2.6, ki nudi storitve za varnost, upravljanje s pomnilnikom in procesi, skrbi za omrežni sklad itd. Jedro deluje kot abstraktna plast med strojno opremo in preostalim programskim skladom.

Izvajalno okolje (ang. Android runtime)

Vključuje set knjižnic, ki zagotavljajo večino funkcionalnosti, ki so na voljo v programskem jeziku Java. Vsaka aplikacija Android teče v svojem procesu, ki ima svoj primerek (ang. instance) navideznega stroja Dalvik VM (ang. Dalvik virtual machine). S tem je vsaka aplikacija izolirana in ne more vplivati na delovanje druge aplikacije. Dalvik VM je napisan tako, da napravi omogoča učinkovito izvajanje več

navideznih strojev brez velike porabe pomnilnika. Aplikacije se zato prevedejo v format .dex (dalvik izvršljiva datoteka). Dalvik VM pa se zanaša na jedro Linux za funkcionalnost, kot sta nitkanje in upravljanje pomnilnika na nižjih ravneh (ang. low-level memory managment).

Knjižnice

Android vsebuje set C/C++ knjižnic, ki uporabljajo različne komponente operacijskega sistema Android. Te knjižnice so na voljo razvijalcu preko Androidovega aplikativnega ogrodja. Seznam knjižnic zajema:

- Sistemske C knjižnice – implementacija jezika C,
- Knjižnice za multimedijo – nudijo podporo za predvajanje in snemanje različnih popularnih avdio in video formatov kot tudi statične slike,
- Opravitelj površine (ang. Surface Manager) – upravlja dostop do podsistema zaslona in neopazno združi 2D in 3D grafične sloje različnih aplikacij,
- 3D knjižnice – implementacija temelji na openGL ES 1.0 API-ju. Knjižnice za izrisovanje uporabljajo tudi strojno opremo (kjer je na voljo),
- LibWebCore – sodoben pogon za brskalnik, ki poganja Androidov brskalnik in aplikacijsko komponento WebView,
- FreeType – izrisovanje fontov v vektorski obliki, in
- SQLite – hitra in zmogljiva relacijska podatkovna baza, ki je na voljo vsem aplikacijam.

Aplikativno ogrodje

Operacijski sistem Android je odprtokodna platforma, kar razvijalcem nudi možnost razvoja bogatih in inovativnih aplikacij. Razvijalci lahko dodobra izkoristijo strojno opremo naprave, med drugim lahko dostopajo do GPS lokacije s pomočjo vgrajenih GPS oddajnikov, nastavljajo alarme, izvajajo storitve v ozadju (ang. Background service) ter skozi aplikacijo dodajajo obvestila v statusno vrstico.

Razvijalci imajo popoln dostop do istih vmesnikov API kot jih uporabljajo sistemske aplikacije. Aplikacijska arhitektura je razvita tako, da poenostavi ponovno uporabo komponent. Vsaka aplikacija objavi svoje sposobnosti in s tem omogoči drugim aplikacijam uporabo teh sposobnosti oziroma zmogljivosti. Isti mehanizmi omogočajo, da lahko uporabnik zamenja osnovne komponente oziroma vgrajene aplikacije.

Aplikativno ogrodje vsebuje različne storitve (ang. Services) in sisteme, kot so na primer:

- Activity Manager – skrbi za življenjski cikel aktivnosti in nudi navigacijski sklad aktivnosti,
- Notification Manager – omogoča aplikacijam prikazovanje različnih sporočil (ang. Alert) v statusni vrstici,
- Resource Manager – nudi dostop do neprogramskih virov aplikacije, kot so na primer lokalizirane besede, grafika in postavitvene datoteke,
- View System – set razredov, izpeljanih iz osnovnega razreda View, pri katerem ima vsak View objekt na voljo pravokotno površino, na katero se izrisuje. Ti objekti predstavljajo gradnike aplikacij, kot na primer seznam, vnosno polje, gumb in slika,
- Ponudniki vsebin – omogoča aplikacijam dostop in delitev svojih podatkov.

Aplikacije

Android vsebuje nabor aplikacij, ki omogočajo uporabo strojne opreme naprave, kot so na primer odjemalec elektronske pošte, program za pisanje, pošiljanje in prejemanje kratkih besednih sporočil (SMS), koledar, navigacija, seznam kontaktov itd. Vse aplikacije so napisane v programskem jeziku Java.

3.2.2 Temelji aplikacije

Android je večuporabniški sistem Linux, kjer ima vsaka aplikacija svojega uporabnika, kateremu sistem dodeli unikatno identifikacijo. Ponavadi vsaka aplikacija teče v svojem Linux procesu, katerega upravlja Android. Vsak proces ima svojo Dalvik VM, tako da se aplikacija izvaja ločeno od drugih aplikacij. S tem nameščena aplikacija živi v svojem varnostnem »peskovniku« (ang. Sandbox). Ko se mora komponenta aplikacije izvesti, Android začne proces, obenem pa ga tudi zaključi, ko le-ta ni več potreben. Na tak način Android implementira načelo najmanjšega privilegija (ang.principle of least privilege), kar pomeni, da ima vsaka aplikacija dostop le do tistih komponent, ki jih potrebuje za opravljanje svojega dela. To ustvari zelo varno okolje, v katerem aplikacije nimajo dostopa do tistih delov sistema, za katere nimajo dovoljenja. Kljub temu operacijski sistem omogoča izmenjavo podatkov med aplikacijami.

3.2.3 Komponente aplikacije

Komponente so ključni gradniki aplikacije opreacijskga sistema Android in skozi njih sistem vstopa v aplikacijo [11]. Vsaka komponenta:

- ni vstopna točka, ampak je lahko odvisna od drugih komponent,
- obstaja kot svoja entiteta in igra specifično vlogo, in
- je edinstven gradnik, ki pomaga definirati celotno obnašanje aplikacije.

Obstajajo štiri vrste komponent, vsaka izmed njih pa ima svoj življenjski cikel, ki definira, kako se ustvari in izniči komponento. Vrste komponent so naslednje:

- **Aktivnost** (ang. Activity)

  Vsaka aktivnost predstavlja en zaslon z uporabniškim vmesnikom. Če vzamemo za primer aplikacijo za prebiranje in pošiljanje elektronske pošte, ima lahko eno aktivnost, ki prikazuje seznam vseh e-poštnih sporočil, drugo aktivnost za sestavljanje e-poštnega sporočila in tretjo aktivnost za prebiranje e-poštnega sporočila. Čeprav aktivnosti delujejo skupaj in tako tudi tvorijo aplikacijo, je vsaka aktivnost neodvisna od ostalih. Tako lahko neka druga aplikacija začne katerokoli od teh aktivnosti (če se seveda prvotna aplikacija strinja).

  Za primer si lahko vzamemo pričetek aktivnosti za sestavljanje e-poštnega sporočila, ki ga prične aplikacija »kamera« in s tem omogoči uporabniku hitro in enostavno pošiljanje pravkar napravljenih slik prek elektronske pošte. To je le eden od načinov uporabe določene aktivnosti posamezne aplikacije s strani neke druge aplikacije.

- **Storitev** (ang. Service)

  Storitev je komponenta, ki teče v ozadju, z namenom izvedbe dolgotrajne operacije ali z namenom izvajanja opravila za oddaljene procese. Storitev ne vsebuje uporabniškega vmesnika. Storitev na primer predvaja glasbo v ozadju, medtem ko uporabnik uporablja neko drugo aplikacijo. Druga komponenta oziroma aktivnost pa lahko zažene storitev in se nanjo priklopi z namenom interakcije s samo storitvijo.

- **Ponudnik vsebin** (ang. Content provider)

  Ponudnik vsebin upravlja s podatki aplikacije, le-te pa se lahko shrani v datotečni sistem, v SQLite podatkovno bazo, na oddaljen strežnik ali katerikoli drug način, ki ga naprava omogoča. Skozi vmesnik ponudnika vsebin lahko aplikacije pridobivajo ali spreminjajo podatke. Uporabni so tudi pri branju in pisanju podatkov, ki so na voljo le specifični aplikaciji.

- **Sprejemnik** (ang. Broadcast receiver)

  Sprejemnik je komponenta, ki se odziva na različne objave. Večina objav izvira iz sistema. Za primer lahko vzamemo obvestilo, da se je zaslon izklopil, da je baterija prazna ali da je bila zajeta slika. Aplikacije lahko tudi same objavljajo obvestila z namenom obveščati ostale aplikacije, da so bili določeni podatki preneseni na napravo in so na voljo. Ne vsebujejo uporabniškega vmesnika, vendar lahko ustvarijo novo obvestilo za uporabnika v statusni vrstici.

Edinstven vidik arhitekture platforme Android je ta, da lahko katerakoli aplikacija zažene komponento druge aplikacije. To v praksi pomeni, da če želimo narediti aplikacijo, pri kateri se s pomočjo video kamere na napravi zajame slika, lahko za ta namen uporabimo že razvito komponento oziroma aktivnost iz druge aplikacije. S tem nam ni treba programirati zajemanja slike, le uporabimo obstoječo komponento, ki to opravlja. Sistem nam omogoča tudi pošiljanje zajete slike po končanju aktivnosti zajemanja nazaj v našo aplikacijo, kjer jo lahko spreminjamo. Uporabniku se bo zdelo, kot da je zajemanje slike del aplikacije, medtem ko je ta aktivnost v resnici del druge aplikacije.

Ko sistem zažene komponento, se začne proces za to aplikacijo (če še ne obstaja) in inicializira potrebne razrede za to komponento. Tako lahko vidimo, da aplikacije Android nimajo ene vstopne točne v aplikacije (ne obstaja metoda main()), kot je to primer pri drugih operacijskih sistemih.

Ker sistem izvaja vsako aplikacijo v svojem procesu in ima omejen dostop do ostalih aplikacij, sama aplikacija ne more neposredno aktivirati komponente druge aplikacije. Zato je potrebno sistemu poslati sporočilo, da želimo izvesti določeno komponento, ki jo sistem nato temu primerno tudi aktivira. Temu sporočilu pravimo »namen« (ang. Intent). To je asinhrono sporočilo in služi kot komunikacijski mehanizem med aplikacijami in sistemom. Od zgoraj naštetih komponent se kar tri aktivirajo z uporabo »namenov«.

3.2.4 Manifest aplikacije

Preden sistem lahko prične z izvedbo posamezne komponente aplikacije, mora preveriti, ali le-ta obstaja. To stori tako, da prebere tako imenovani manifest aplikacije, ki obstaja v obliki XML datoteke, v kateri aplikacija definira:

- vse svoje komponente,
- verzijo aplikacije,
- minimalno verzijo Androida, in
- uporabljene strojne in programske značilnosti naprave.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          package="com.appes.creativity"
          android:versionCode="1"
          android:versionName="1.0">
   <uses-sdk android:minSdkVersion="4" />
   <application
                android:icon="@drawable/icon"
                android:label="@string/app_name" >
          <activity
                android:name=".MainMenuActivity"
                android:screenOrientation="landscape"
                android:label="MojaAplikacija">
                <intent-filter>
                      <action android:name="android.intent.action.MAIN" />
                      <category android:name="android.intent.category.LAUNCHER" />
                </intent-filter>
          </activity>
          <activity
                android:name=".GameActivity"
                android:screenOrientation="landscape"
                android:label="@string/app_name" >
          </activity>
   </application>
</manifest>
```

Zgoraj navedeni manifest opisuje aplikacijo »Moja Aplikacija« različice 1.0, ki deluje na vseh napravah z Androidom od verzije 1.6 dalje. Vsebuje dve aktivnosti, prva služi kot vstopna točka, druga pa je odvisna od prve. Vsaka aktivnost ima definirano tudi svoje prikazno ime, orientacijo zaslona in ime razreda aktivnosti. Razred aktivnosti je izpeljan iz razreda Activity.

3.2.5 Viri aplikacije

Androidova aplikacija ni sestavljena le iz programske kode, pač pa so potrebni tudi drugi viri, ko so na primer slike, zvočni zapisi ali karkoli drugega, kar je povezano z vizualno predstavitvijo aplikacije. Sem spadajo tudi animacije, različni uporabniški vmesniki, lokalizirane vrednosti besed, uporabljenih v aplikaciji itd.

Android povezuje vse vire aplikacije s preprostimi XML datotekami, kar omogoča preprosto prilagajanje aplikacije različnim napravam in njenim strojnim sposobnostim. Aplikacije vsebujejo mehanizme za prilaganje uporabniških vmesnikov glede na velikost zaslona in kakovost zaslona (DPI), podpore za večjezičnost, animacije in podobno. Aplikacijo lahko prikažemo drugače na napravah, ki imajo različne zaslone, tako da samo definiramo dve različni XML datoteki in jima v ime dodamo ustrezno oznako (ldpi, mdpi, hdpi, xhdpi itd.). Oznake se razlikujejo glede na mehanizem, lahko pa tudi sami definiramo svoj mehanizem oziroma pravila. Vizualni del aplikacije lahko sprogramiramo ali pa ga definiramo v XML datoteki.

3.2.6 Aktivnost

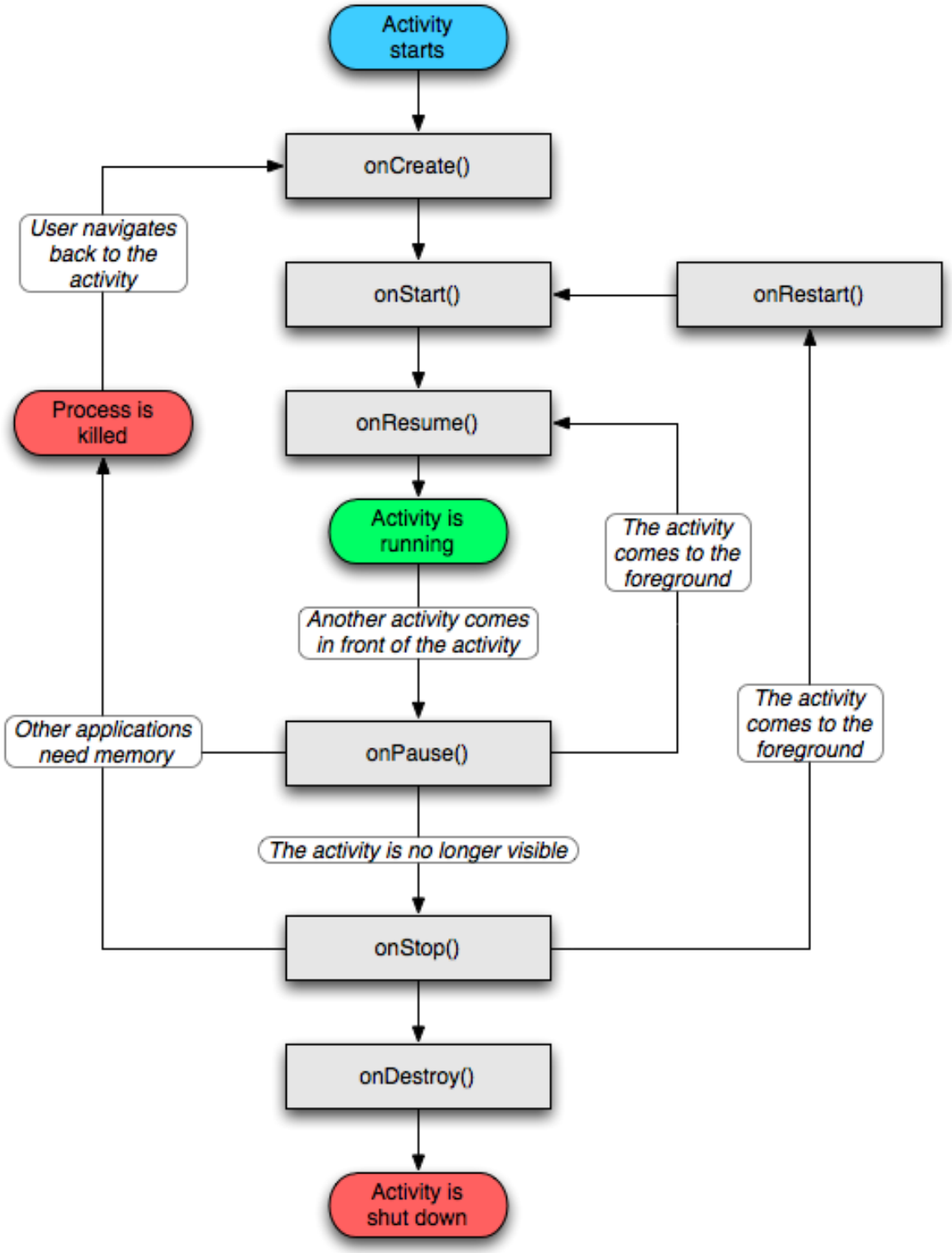
Aktivnost je aplikacijska komponenta Androida, ki nudi zaslon, s katerim uporabnik komunicira. Vsaka aktivnost dobi okno, na katerega se izrisuje uporabniški vmesnik. Okno ponavadi zapolni celoten zaslon, vendar je lahko tudi manjši in »plava« na vrhu ostalih oken.

Aplikacijo ponavadi sestavlja več aktivnosti, ki delujejo kot celota. Običajno ena aktivnost služi kot vstopna točka, ki se pojavi uporabniku ob prvem zagonu aplikacije. Vsaka aktivnost lahko tudi začne neko drugo aktivnost. Vsakič, ko se nova aktivnost prične, se predhodna aktivnost ustavi. Zato so aktivnosti zložene skladno, in ko se ena izmed njih prične, se doda na vrh sklada. To pomeni da se uporablja princip LIFO (ang. last in first out). Na ta način je uporabniku omogočen povratek na predhodno aktivnost s pritiskom ustrezne tipke.

Uporabniški vmesnik aktivnosti predstavlja hierarhijo objektov, izpeljanih iz razreda »View«. Vsak »view« predstavlja pravokotno površino znotraj okna aktivnosti, s katero trenutno upravljamo. »View« je lahko gumb, slika, vnosno polje, forma ali nekaj drugega. Poleg tega Android nudi številne vnaprej določene razrede, izpeljane iz »View« razreda, ki nudijo različne oblike delovanja, odvisno od naših zahtev.

Ker se aktivnost lahko kadarkoli ustavi ali zaključi, moramo za uspešno implementiranje aplikacijo poznati njen življenjski cikel. V ta namen razred Activity vsebuje metode, ki se uporabijo ob različnih dogodkih v zvezi z aktivnostjo.

Slika 12 [11] prikazuje celoten življenjski cikel aktivnosti in klice ustreznih metod v povezavi z dogodki cikla.



Slika 12: Življenjski cikel Aktivnosti.

Kot vidimo s slike (slika 12), se lahko aplikacija med izvajanjam večkrat ustavi ali celo uniči oziroma zaključi. Zato Androidov aplikativni model temelji na sprotnem shranjevanju podatkov. Zaradi tega moramo vsakič, ko se izvajanje aktivnosti konča oziroma se izvede metoda *onStop(),* shraniti vse delo uporabnika.

3.3 Android SDK

Android SDK (ang. Software development kit) je programska oprema za razvoj aplikacij za platformo Android, ki je odgovorna za pripravo programske kode aplikacije, skupaj z vsemi potrebnimi podatki in viri aplikacije. Rezultat priprave je tako imenovani Android paket oziroma arhivska datoteka s končnico .apk. Vsa koda in vsi podatki so shranjeni v eni .apk datoteki, ki se uporabi za namestitev na napravo [10,11].

Android SDK vsebuje:

- primerke projektov in njihovo programsko kodo,
- razvijalska orodja,
- emulator, in
- potrebne knjižnice za pripravo in gradnjo aplikacij.

3.3.1 Emulator

Android SDK vsebuje navidezni emulator mobilne naprave, ki se izvaja na osebnem računalniku (slika 13 [11]). Omogoča nam preizkušanje, izdelavo prototipa in razvoj aplikacij za Android brez fizične naprave.

Slika 13: emulator Android naprave.

Deluje tako, da simulira strojno in programsko opremo fizične naprave, s to razliko, da ne moramo opravljati klicev, lahko jih le simuliramo. Emulator vsebuje različne tipke, ki jih najdemo na različnih napravah, medtem ko emulator upravljamo z miško.

Za lažje testiranje aplikacij emulator uporablja konfiguracije AVD (Android virtual device). AVD omogočajo enostavno spreminjanje različnih kombinacij strojnih in programskih karakteristik emulirane naprave.

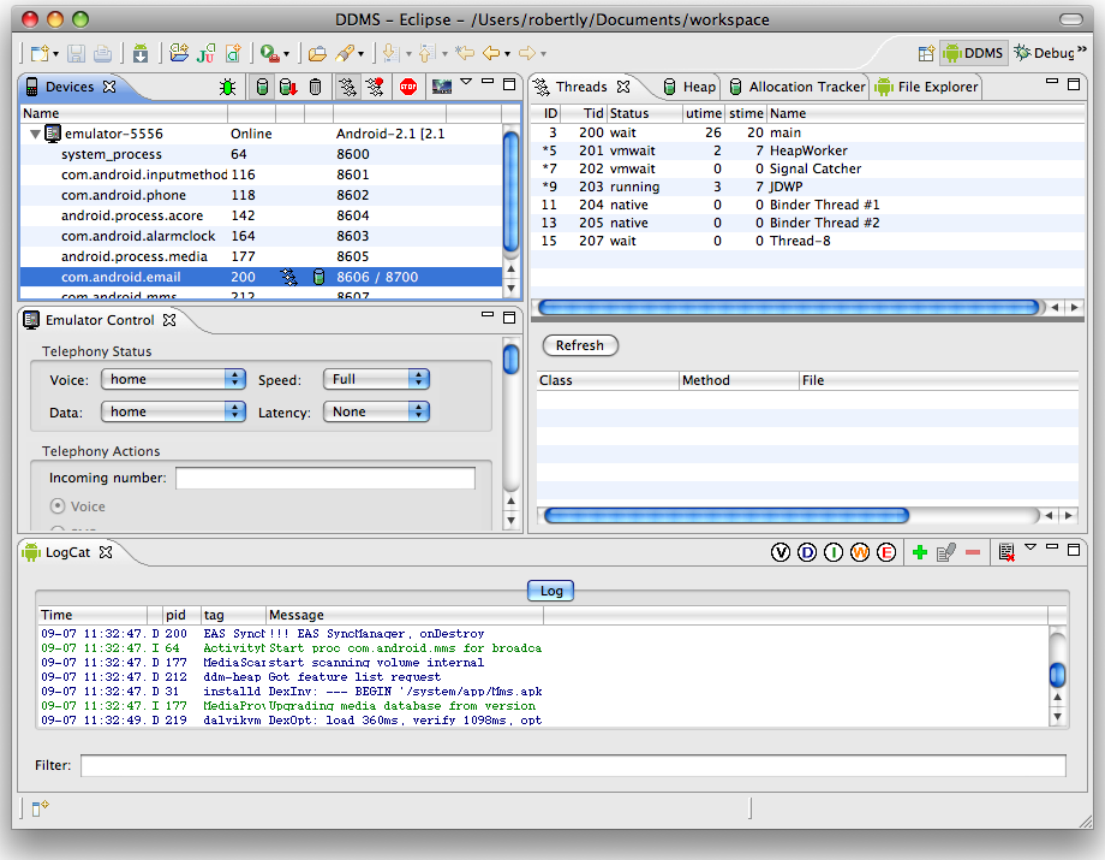
S tem si lahko ustvarimo več različnih AVD konfiguracij in opravimo preizkuse aplikacije na različnih velikostih in kakovostih zaslona. Emulator vsebuje različna razhroščevalna orodja, kot so izpis celotnega dnevnika in simulacijo prekinitev (ko naprava prejme SMS ali klic).

3.3.2 Razhroščevanje

Ker se pri vsakem razvoju rado zatakne, nam Android SDK nudi različna orodja za razhroščevanje naše programske kode [11].

Nekatera orodja za razhroščevanje so:

- **ADB**: se obnaša kot posrednik med napravo in razvojnim okoljem (PC). Nudi storitve, kot so prenos podatkov med napravo in razvojnim okoljem, komunikacijo z okoljem unix na napravi,
- **DDMS**: grafični program, ki komunicira z napravo preko orodja ADB. DDMS lahko zajame sliko iz naprave, zbere podatke o nitkanju in skladu, prikazuje alokacije pomnilnika in podobno (slika 14 [11]).
- **JDWP**: virtualna naprava Dalvik VM podpira protokol JDWP, prek katerega se razhroščevalniki pripnejo na navidezni stroj. S tem dobimo vpogled v delovanje navideznega stroja.

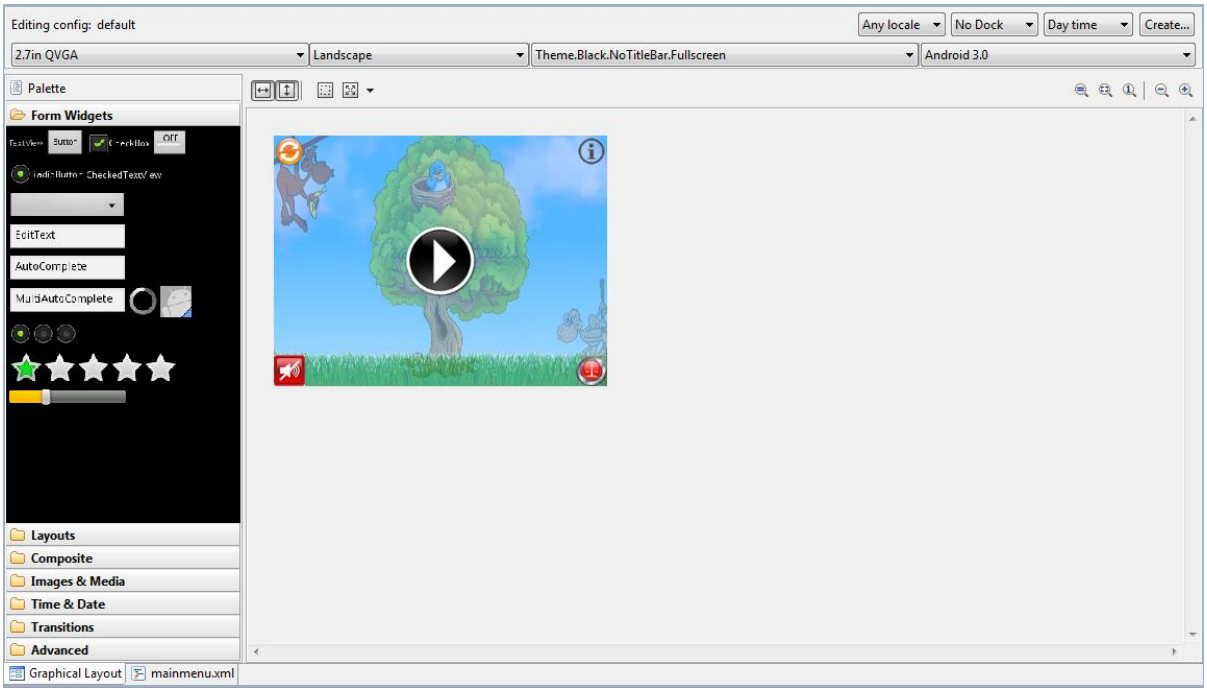


Slika 14: Razhroščevalnik DDMS.

3.4 Razvojno okolje

Ker Android SDK nudi vse, kar je potrebno za razvoj Android aplikacij, potrebujemo še razvojno okolje IDE, ki zna uporabljati Android SDK in omogoča pisanje programske kode zanj.

Najbolj priljubljena IDE-ja sta Netbeans IDE in Eclipse IDE. Oba dopuščata možnost razvoja projektov oziroma aplikacij za platformo Android. Glavna razlika med njima je ta, da Eclipse vsebuje tudi vtičnik (ang. plugin) – razvojna orodja ADT (ang. Android development tools) [13]. Vtičnik razširi delovanje razvojnega okolja in omogoča razvijalcu preprosto izvajanje, razhroščevanje in emularanje programske kode. Orodja nudijo tudi opcijo za vizualno urejanje in izdelavo uporabniških vmesnikov (slika 15).



Slika 15: Vizualni urejevalnik uporabniškega vmesnika.

4 Razvoj igre

4.1 Creativity

Za predstavitev praktičnega dela diplome smo razvili igro po imenu Creativity. Creativity je družabna igra za mobilne naprave. Pred začetkom igre se igralci razdelijo v ekipe. Za igranje igre potrebujemo najmanj dve ekipi igralcev, največje število ekip pa je šest. Število igralcev na ekipo ni omejeno. Nato se določi vrstni red ekip, ki vpliva na to, katera ekipa bo kdaj začela.

Cilj igre je zaključiti igralno pot pred ostalimi ekipami z ugibanjem besede, ki jo naključno določi sama igra. Ekipa, ki je na vrsti, določi igralca, ki bo predstavljal ekipo v trenutnem krogu. Igralec nato dobi besedo, ki jo mora predstaviti z govorom, risanjem ali mimiko ostalim članom svoje ekipe. Če ostali člani ugotovijo besedo v določenem času, napredujejo po igralni plošči.

Ključne sestavine igre Creativiy so:

- **igralna površina**, ki predstavlja igralno ploščo, po kateri se ekipe oziroma glavni liki ekip premikajo (slika 17),
- **glavni liki**, ki predstavljajo vsako ekipo, in so ločeni med seboj glede na barvo oziroma oznako ekipe kateri pripadajo,
- **kolo sreče**, ki ima 24 polj. Vsako polje vsebuje številko od 1 do 3, ki predstavlja število premikov, ki jih lahko uporabnik naredi, in simbol za način predstavitve besede – mimika, govor ali risanje, in
- **nabor besed oziroma slovar** (ang. Dictionary), iz katerega izhajajo besede oziroma igralne naloge,
- **vstopni uporabniški vmesnik**, kjer uporabnik nastavlja parametre igre, kot so na primer zvok, izbira igralne plošče, izbira težavnosti (slika 16).

Slika 16: Vstopni uporabniški vmesnik igre Creativity.

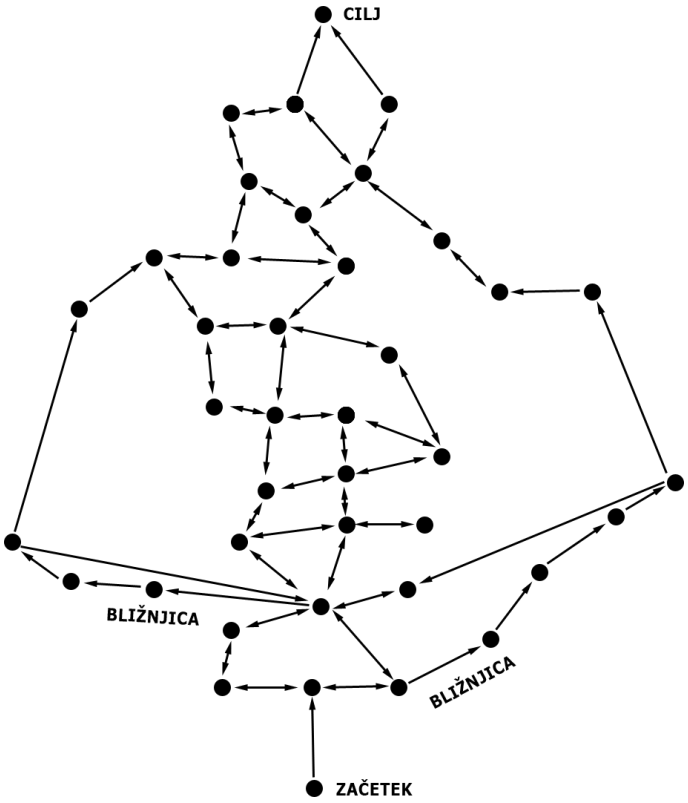
Igra poteka izmenično, v vsakem krogu je na vrsti ena ekipa. Ko ekipa krog konča, je na vrsti naslednja. Igra se konča, ko zmagovalna ekipa zaključi pot po igralni plošči.

Igra vsebuje še različne zaplete, ki jo dodatno popestrijo. Zapleti oziroma posebne funkcije v igri so:

- štiri polja kolesa sreče so posebna polja:
    - na dveh poljih se skriva posebno vprašanje, na katerega imajo možnost odgovarjati vse ekipe, zmagovalec posebnega vprašanja pa dobi štiri točke oziroma premike po igralni plošči,
    - na enem polju dobi ekipa, ki je na vrsti, v primeru pravilnega odgovora dodatno potezo oziroma so na potezi še enkrat in lahko znova zavrtijo kolo sreče
    - na enem polju je skrit negativen zaplet, saj je ekipa prisiljena počakati en krog.
- Na igralni plošči so tudi tako imenovane bližnjice. Ob prihodu na polje, ki označuje bližnjico, dobi ekipa dodatno vprašanje in v primeru pravilnega odgovora napreduje po bližnjici naprej. V nasprotnem primeru jo igralna plošča vrne nazaj na osrednjo pot. Bližnjice so postavljene na takšen način, da zahtevajo od ekipe določeno mero tveganja, saj se nahajajo izven osrednje poti. V primeru, da se tveganje obrestuje, pa se ekipe hitreje povzpnejo po igralni plošči.

Igralna plošča je povezan graf, ki je sestavljen iz točk oziroma polj (slika 13). Vsako polje je povezano z vsaj dvema drugima poljema. Iz vsake točke na grafu je mogoča pot do katerekoli druge točke na grafu. Skozi graf poteka osrednja pot do cilja in več

stranskih poti oziroma bližnjic. Bližnjice so enosmerne povezave, kar pomeni, da ni mogoč povratek igralnega lika po bližnjici navzdol.



Slika 17: Graf igralne plošče igre Creativiy.

Igralni lik ekipe se premika po povezavah med posameznimi polji na igralni plošči. Smer možnega premika se uporabniku izriše kot puščica, ki utripa nad poljem. V primeru pritiska na polje ali puščico se lik temu ustrezno premakne do izbranega polja. Ko ena izmed ekip doseže ciljno polje, se igra zaključi.

4.2 Zahteve igre Creativity

Ob načrtovanju izdelave smo sprva definirali osnovne zahteve, ki naj bi jih igra izpolnjevala. Potrebna je bila torej specifikacija strojne in programske opreme, poleg tega pa je bilo potrebno definirati splošno igralnost, kar bi se odražalo na zadovoljstvu uporabnikov.

Zahteve s strani uporabnika in strojne te programske kode so naslednje:

- igra mora delovati tekoče na vseh mobilnih napravah z operacijskim sistemom Android (na vseh aktualnih različicah),
- igra se mora prilagajati na vse različne velikosti in kakovosti zaslonov mobilnih naprav,
- uporabnik mora z igro upravljati preko zaslona na dotik,
- igra mora vsebovati nabor besed oziroma slovar (ang. Dictionary), do katerega lahko dostopamo ter ga po potrebi spreminjamo,
- igra mora imeti opcijo za posodabljanje nabora besed preko povezave s strežnikom,
- nabor besed mora biti shranjen lokalno, torej na sami napravi,
- igralec lahko sam doda svoj nabor besed in ga uporabi v igri,
- vsaka beseda, ki je del osnovnega nabora, mora vsebovati atribute, ki določajo, na kakšen način se lahko beseda predstavi oziroma primeren način ponazarjanja iskane besede (mimika, govor, risanje),
- vsaka beseda mora biti kategorizirana glede na zvrst (šport, splošno, medicina, in podobno) in težavnost (lahko, srednje, težko),
- igra mora podpirati več različnih igralnih plošč, ki se razlikujejo po postavitvi točk in povezav,
- igra mora podpirati posodabljanje igralnih plošč s strežnikom,
- igra mora podpirati igranje na več napravah hkrati – preko povezave bluetooth ali brezžičnega omrežja,
- kolo sreče mora delovati na podlagi hitrosti zasuka in mora implementirati fizične zakone vrtenja kroga okoli osi,
- igralna plošča mora biti uporabniku prikazana v 2D prostoru (pogled s strani),
- igralna plošča vedno presega velikost zaslona mobilne naprave, zato je potrebno uporabniku omogočiti premikanje pogleda po plošči in povečevanje/pomanjševanje plošče.
- igralna površina mora vsebovati ozadje, ki je odvisno od igralne plošče, in se izrisuje glede na trenutno pozicijo,
- Igralni liki ekip morajo vsebovati animacije, ki se uporabijo glede na stanje ekipe:
    - animacija v primeru zmage in poraza ekipe

  - animacija v primeru nepravilnega oziroma pravilnega odgovora
  - animacija za plezanje po povezavah med točkami
  - animacija za premikanje po točki v levo in desno
- Igra mora poleg animacij predvajati tudi zvoke,
- Igra potrebuje zaslon s tako imenovanim pojavnim oknom (ang. popup) oknom, ki služi obveščanju uporabnika. Okno mora poleg sporočila vsebovati tudi gumbe kot na primer v redu, zapri, naprej, razveljavi,
- Igra potrebuje izpis vseh ekip in njihov vrstni red. Ob zamenjavi kroga se mora vrstni red ekip posodobiti in s tem tudi animirati.

4.3 Izdelava iger v operacijskem sistemu Android

Prva naloga pri izdelavi igre Creativity je bila ugotoviti, kako izdelati preprosto aplikacijo v operacijskem sistemu Android, implementirati njeno delovanje, opisano v 3. poglavju, in jo predelati v mobilno igro z implementacijo vseh pravil, potrebnih za delovanje igre na način, kot je opisan v 2. poglavju.

Za izdelavo kakovostnih iger v Androidu je podjetje Google izdalo različne primerke programske kode ter različne članke in predavanja o izdelavi le-teh. Primerki vsebujejo različne preproste igre, ki morebitnim razvijalcem služijo kot oporne točke.

Eno izmed najbolj uporabnih predavanj ima naslov »Writing Real Time Games for Android« [9,12], na katerem so predstavljeni vsi načini izdelave igre. Poleg tega dobimo tudi informacije o tem, na kaj moramo biti najbolj pozorni pri razvoju igre.

4.3.1 Optimizacija iger

Ker so aplikacije operacijskega sistema Android napisane v programskem jeziku Java, imajo zaradi tega določene pomankljivosti v primerjavi s tistimi, ki so spisane v programskem jeziku C++. Med pomanjkljivosti vsekakor spada delo s pomnilnikom. Java dodeljuje pomnilniški prostor, kjer tega običajno ne bi pričakovali, kot na primer:

canvas.drawText('To je kratek text',x,y, null);

Arrays.sort()

Class.getClassName()

Iterator

HashMap

V vseh zgornjih primerih Java ob vsakem klicu dodeli nov pomnilniški prostor, kar privede do sprostitve pomnilniškega prostora (ang. garbage collector).

Poleg dela s pomnilnikom je potrebno tudi vedeti, da so klici metod v Javi lahko časovno potratni, še posebej takrat, če se ena in ista metoda kliče po več stokrat na vsak okvir. Da se temu izognemo in stvar pohitrimo, uporabimo statične metode in naredimo atribute javne. To je bolj,e kot da opredelimo za vsak atribut metodo »get in set«. V ta namen v programski kodi uporabimo ključno besedo »final«. Klici metod skozi vmesnike (ang. interface) so približno 30 % počasnejši kot preko navadnega klica.

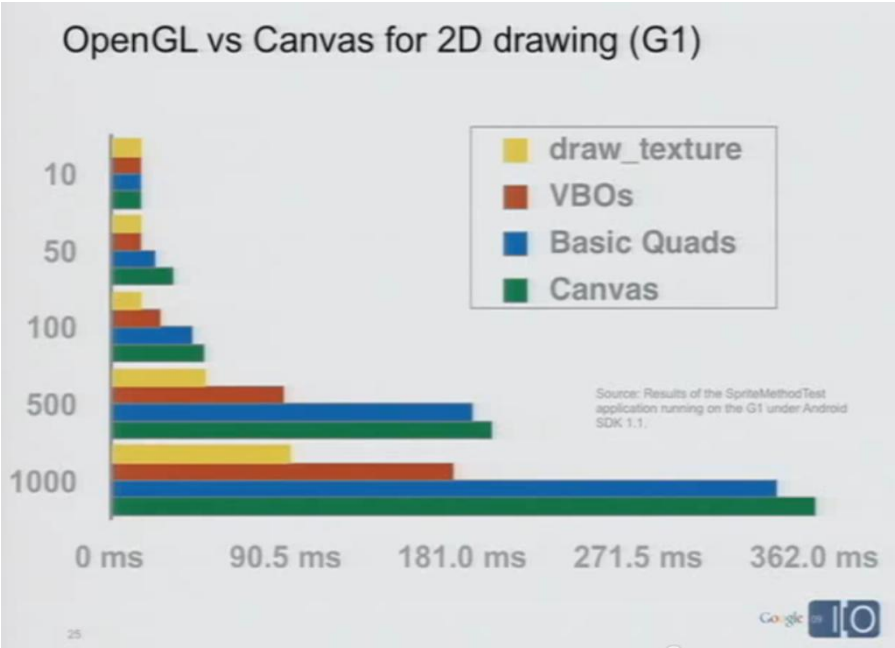
Ker so mobilne naprave majhne, vse ne vsebujejo enote FPU (ang. floating point unit) [15], ki je odgovorna za preračunavanje necelih števil oziroma števil z vejico (ang. float). Če naprava nima enote FPU, je zato celotno računanje izvedeno programsko, kar pomeni, da je računanje z necelimi števili počasno. Ta problem lahko rešimo tako, da vpeljemo tako imenovani Fixed point.

4.3.2 Izrisevanje v Androidu

V Androidu je možno izrisovanje na zaslon na dva načina:

- Z uporabo »**canvas-a«,** ki se izvaja na procesorju (CPU) - ta način se uporabi v primeru preprostih iger, ko ni potrebno veliko animacij ali izrisovanj, in
- Z uporabo **openGL ES**, ki je lahko podprto s strojno opremo, drugače je realiziran s pomočjo programske opreme. Na ta način je možno izrisovanje v 2D in 3D prostoru. V openGL-u. obstaja več tehnik izrisevanja

Slika 18 [14] opisuje primerjavo hitrosti izrisa številnih elementov med canvas in openGL sistemom. Kot je razvidno s slike, je sistem openGL hitrejši kot Canvas, ko imamo na zaslonu več kot 50 elementov.



Slika 18: Primerjava hitrosti izrisevanja Canvas sistem v primerjavi z open GL sistemom.

Z uporabo sistema Canvas lahko igro implemenitramo na dva različna načina:

1. S pomočjo Androidovih aplikacijskih elementov **View**, pri katerih ni potrebno skrbeti o tem, kako se bodo objekti izrisovali in animirali, temveč je potrebno samo definirati, kaj se mora izrisati. Za to poskrbi Androidova hierhija View objektov. Ta način je dober za razvoj iger s počasnim osveževanjem, statično grafiko in vnaprej določenimi animacijami. Slabost aplikacijskega elementa View pa je, da ni prilagojen za hitro izvajanje igre.

2. Če potrebujemo hitrejše osveževanje, uporabimo **SurfaceView**, ki je posebna razširitev View objekta. Omogoča namensko površino (ang. dedicated surface) za risanje. SurfaceView omogoča, da se izrisovanje na zaslon dogaja v drugi niti (ang. Thread), kar pomeni, da aplikaciji ni potrebno čakati na sistemsko hierarhijo View objektov za izris.

4.4 Pozdravljen, svet

Kot pri vsakem novem programskem sistemu oziroma jeziku smo se lotili izdelave tradicionalne igre »pozdravljen, svet« (ang. »hello world«) s pomočjo Canvas sistema.

V 2. poglavju smo omenili, da moramo za delujočo igro implementirati zanko programske kode (»gameloop«). Da lahko igro prikažemo, potrebujemo komponento Android Activity, v kateri prikažemo uporabniški vmesnik in zaznavamo dogodke, ki se dogajajo z našo aktivnostjo.

Za izdelavo preproste pozdravljen, svet igre smo potrebovali razred (GameActivity), ki je razširjen iz razreda aktivnost, razred (GameCanvas), ki je razširjen iz razreda SurfaceView ter nit (GameThread), ki je odgovorna za izrisovanje na SurfaceView.

Spodaj je programska koda aktivnosti igre, ki predstavlja vstopno točko v igro.

```
public class GameActivity extends Activity {
    private GameThread gameThread = null;

        public void onCreate(Bundle savedInstanceState) {
           super.onCreate(savedInstanceState);
           setContentView(R.layout.game);
           gameThread = ((GameCanvas)findViewById(R.id.gamecanvas)).getThread();
         }
         protected void onPause() {
                  super.onPause();
                  gameThread.onPause();
         }
         protected void onResume() {
                  super.onResume();
                  gameThread.onResume();
         }

         public void onStop() {
                  super.onStop();
                  gameThread.onStop();
         }
}
```

Zgoraj navedena programska koda naloži datoteko XML (ki se nahaja v /res/game.xml), v kateri je definiran uporabniški vmesnik za igro. S tem se ustvari nov objekt GameCanvas, ki razširja razred SurfaceView. GameCanvas vsebuje nit, v kateri se izvaja izrisovanje igre, zato shranimo referenco niti, da bi jo lahko obveščali o dogodkih v zvezi z življenjskim ciklom aktivnosti.

Spodaj je programska koda razreda GameCanvas, ki predstavlja risalno površino igre.

```
class GameCanvas extends SurfaceView implements SurfaceHolder.Callback {

   private Context context;
   private GameThread thread;
   public static int x = 0;
         private long mLastTouchTime = 0;

   public GameCanvas(Context context, AttributeSet attrs) {
      super(context, attrs);
      SurfaceHolder holder = getHolder();
      holder.addCallback(this);
      this.context = context;
      thread = new GameThread(holder,context);
      setFocusable(true);
   }

   public GameThread getThread() {
         return this.thread;
   }

   public void surfaceChanged(SurfaceHolder holder, int format, int width, int height) {
      thread.onSurfaceChange(width, height);
   }

   public void surfaceCreated(SurfaceHolder holder) {
         if(!thread.isAlive())
                  thread = new GameThread(holder,context);
         thread.onStart(getWidth(),getHeight());
      thread.setRunning(true);
      thread.start();
   }

         public void surfaceDestroyed(SurfaceHolder holder) {
      boolean retry = true;
      thread.setRunning(false);
      while (retry) {
         try {
            thread.join();
            retry = false;
         } catch (InterruptedException e) {}
      }
   }
}
```

Zgornja programska koda predstavlja razred, razširjen iz razreda SurfaceView. Da lahko dogodke v povezavi s SurfaceView-om zaznavamo med igro, mora razred GameSurface implementirati razred SurfaceHolder.Callback. Ta razred nam s tem omogoča zaznavanje sprememb risalne površine. Zato moramo implementirati 3 metode:

- surfaceCreated – metoda se pokliče, ko je površina ustvarjena,
- surfaceChanged – metoda se pokliče, ko se velikost površine spremeni in
- surfaceDestroyed – metoda se pokliče, preden se površina uniči.

Ker SurfaceView služi kot risalna površina, potrebujemo nit (ang. Thread), ki bo izrisovala elemente igre na zaslon ter ne bo onemogočala glavne niti, v kateri se odvija naša aktivnost. Zato v konstruktorju razreda GameSurface ustvarimo novo nit, ki jo začnemo izvajati, ko se površina ustvari (Thread.start()). Če bi nit začeli izvajati v konstruktorju, bi prišlo do napake (ang. Exception), saj bi začeli izrisevati na površino, ki še ni ustvarjena. To bo pripeljalo do sistemskega sporočila ANR (ang. Application not responding), ki pomeni, da se naša igra ne odziva. Tu moramo biti še posebej pozorni na pravilno implementacijo dogodkov v povezavi z aktivnostjo. V primeru, da uporabnik zapusti aplikacijo in se po določenem času vrne, se bo metoda surfaceCreated poklicala dvakrat konstruktor pa le enkrat . To pomeni, da bomo v času drugega klica surfaceCreated želeli pričeti izvajati nit, ki se je že končala izvajati. Temu se lahko izognemo z zaznavanjem, ali je naša nit še »živa« (ang. Is Alive).

Spodnji zapis programske kode predstavlja metodo »run« razreda GameThread, ki je razširjen iz razreda Thread. Ta metoda predstavlja dejansko zanko programske kode (»gameloop«), kjer se izrisuje celotna igra. Ta gameloop je najpreprostejše oblike, saj deluje na vsaki strojni opremi drugače – na hitrejši se izvede večkrat, na počasnejših pa manjkrat. Zgornja zanka izriše le text »Pozdravljen, svet« in ne vsebuje posodabljanja stanja igre, ker za to v tem primeru ni potrebe.

```
void run() {
   final string str = 'Pozdravljen svet';
   final Paint p = new Paint(Color.WHITE);
    while(running) {
          try {
             Canvas c = holder.lockCanvas(null);
             c.drawText(str,50,50,p);
          } finally {
              If(c != null) holder.unlockCanvasAndPost(c);
          }
   }
}
```

4.5 Aktivnosti igre Creativity

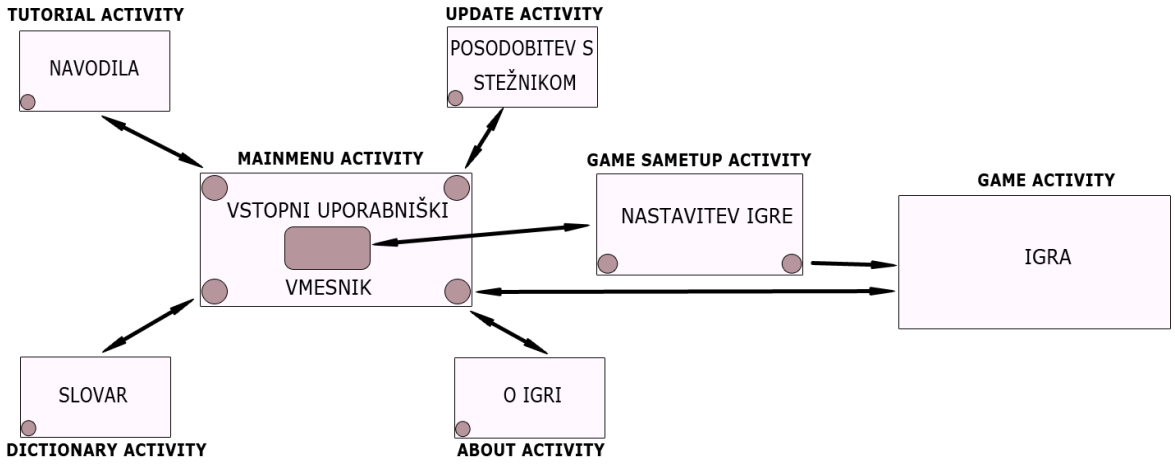
Ko smo zaključil z izdelavo osnovne igre »podravljen, svet« smo nadaljevali z implementacijo ostalih aktivnosti, ki tvorijo igro Creativity.

Igra je sestavljena iz več zaslonov oziroma objektov, izpeljanih iz razreda Aktivnost. Skupaj tvorijo celoten uporabniški vmesnik igre kot tudi zaslon, kjer se igra odvija.

V uporabniškem vmesniku lahko uporabnik:

- nastavi parametre igre (GameSetupActivity),
- izklopi zvok in dostope do ostalih »podmenijev« (MainMenuActivity),
- ureja nabor besed (DictionaryActivity),
- posodobi nabor besed in igralnih plošč preko oddaljenega strežnika (UpdateActivity) in
- prebere navodila (TutorialActivity) ali informacije o igri (AboutActivity).

Slika 19 predstavlja povezanost aktivnosti v igri Creativity. Vsaka povezava ponazarja prehod iz ene aktivnosti v drugo in smer možnega prehoda.



Slika 19: Povezanosti aktivnosti igre Creativity.

Uporabniški vmesnik aktivnosti ponazori s preprosto datoteko XML. V nadaljevanju sta zapisana in podrobno razložena dva primera XML datoteke:

Primer 1:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="@drawable/menu_background">
        <ImageView
                android:id="@+id/update"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:background="@drawable/update"
                android:layout_alignParentLeft="true"
                android:layout_alignParentTop="true"
                android:onClick="onButtonPress"
         />
         <Button
                android:id="@+id/play"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_centerHorizontal="true"
                android:layout_centerVertical="true"
                android:background="@drawable/play"
                android:onClick="onButtonPress"
          />
</RelativeLayout>
```

Primer 1 vsebuje dva objekta razreda View, to sta ImageView in Button, ter en objekt tipa ViewGroup, to je RelativeLayout. Vsi trije objekti imajo svoje atribute, ki opisujejo njihovo velikost, postavitev na zaslonu in delovanje.

Primer 2:

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent">
        <view xmlns:android="http://schemas.android.com/apk/res/android"
                class="com.appes.creativity.GameCanvas"
                android:id="@+id/gamecanvas"
                android:layout_width="fill_parent"
                android:layout_height="fill_parent"
          />
</FrameLayout>
```

Primer 2 vsebuje en objekt razreda ViewGroup, to je FrameLayout, in en objekt razširjen iz razreda View, to je GameCanvas. Razlika med tem in prvim primerom je ta, da je v drugem primeru uporabljen objekt, razširjen iz objekta View, ki ni iz zbirke Androidovih vnaprej definiranih elementov, temveč smo ga napisali sami ter definirali njegovo delovanje.

Oba primera na začetku uporabljata XML vozlišče, ki predstavlja razred, izpeljan iz razreda ViewGroup, ta pa vsebuje različne razrede, izpeljane iz objekta View. Tako na primer razred Button v Androidu predstavlja izpeljanko iz razreda View in služi kot gumb na zaslonu, na katerega uporabnik lahko pritisne, ImageView pa deluje kot razred za predstavitev slike na zaslonu. Te objekte lahko tudi sami razširjamo in posodabljamo njihovo delovanje [10,11].

ViewGroup je razred, ki predstavlja skupino objektov, izpeljanih iz razreda View in definira, kako bodo ti objekti postavljeni na zaslon. Android nudi več različnih vnaprej definiranih razredov, razširjeni iz objekta ViewGroup, ki se razlikejo po načinu predstavitve svojih objektov.

Najpogostejši razredi, izpeljani iz razreda ViewGroup:

- relative layout,
- linear layout,
- table layout,
- grid view in
- list view.

Odvisno od potrebe igre oziroma aplikacije uporabimo primerno razporeditev elementov. Vsak razpored je treba definirati v določeni mapi projekta Android (res/layout). Da v programski kodi predstavimo uporabniški vmesnik s pomočjo XML datoteke, moramo najprej postaviti XML datoteko v primerno mapo, nato pa lahko v programski kodi uporabimo spodnjo vrstico kode:

```
setContentView(R.layout.game);
```

V tem primeru je *R.layout.game* konstanta, ki jo ustvari oziroma generira Android. Ta naloži in ustvari objekte iz XML datoteke »game«, ki se nahaja v /res/layout/game.xml.

Do objektov v XML datoteki lahko dostopamo tudi iz programske kode, in sicer tako, da želenemu objektu dodamo atribut »android:id«. S tem lahko v programski kodi dostopamo do objekta, ki je predstavljen z XML datoteko, in sicer s pomočjo spodnje vrstice kode:

```
MyView myView = (MyView)findViewById(R.id.myview);
```

V tem primeru je R.id.myview konstanta, ki jo generira Android, in je v XML datoteki predstavljena z XML atributom »android:id«='myview'. Na ta način lahko preprosto manipuliramo vse objekte in ni treba programirati uporabniškega vmesnika.

4.6 Konsistentnost igre

S tem ko smo obdelali celotni potek aktivnosti, se lahko vrnemo na razvoj zanke in logike igre. Da bi igra potekala na vseh napravah konsistentno, smo za implementacijo zanke uporabili metodo »Konstantna hitrost igre neodvisna od FPS«, opisano v 2. poglavju.

Zanko iz primera »Pozdravljen, svet« smo prilagodili tej metodi in dodali posodabljanje stanja igre. Želeli smo ugotoviti, ali metoda deluje konsistentno na vseh napravah, saj je emulator počasen in neprimerljiv s fizično napravo, poleg tega pa smo želeli ugotoviti tudi, ali interpolacija vpliva na vizualno predstavo igre. To smo dosegli tako, da smo v igro vpeljali objekt, ki se je z različnimi hitrostmi premikal po zaslonu, nato pa smo izmeril čas, v katerem je objekt prepotoval določeno razdaljo.

Ta preizkus smo ponovili na različno zmogljivih napravah:

- Samsung galaxy S
- Sony Ericsson XPERIA x8
- Samsung Spica
- HTC Tatu

Igra se je izkazala za konsistentno na vseh napravah. Stanje igre se je posodabljalo 25-krat na sekundo, izrisovanje pa je potekalo tako hitro kot je omogočala naprava, od 40–60-krat na sekundo. Namen preizkusov je ugotoviti, ali prihaja pri uporabi interpolacije do razlik v vizualni predstavi igre na različnih napravah. Izkazalo se je, da pri hitro premikajočih se objektih z interpolacije dobimo takšen učinek, kot da igra deluje hitreje od 25 UPS.

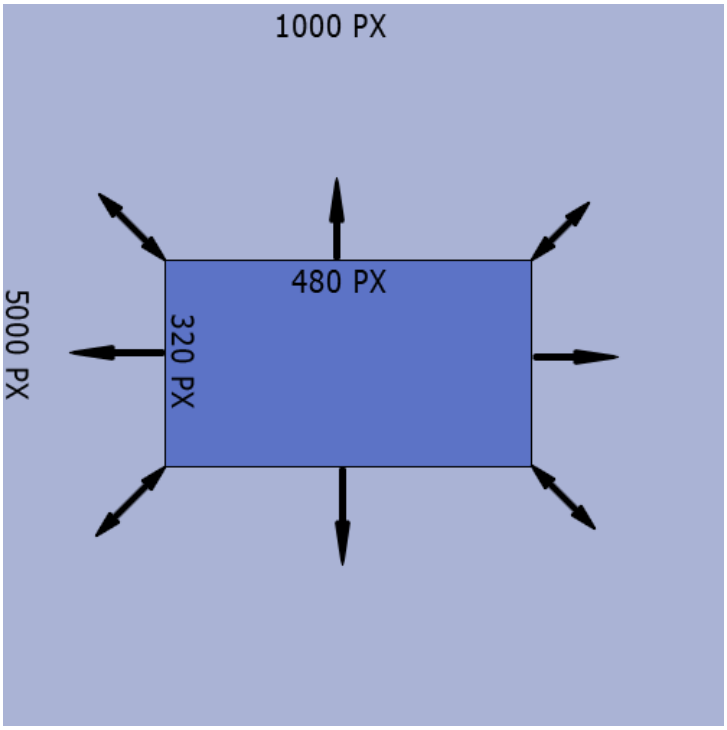
Ker so bili rezultati testiranja konsistenti, smo razvoj nadaljevali z implementacijo logike igre.

4.7 Igralna površina

Ker je igralna površina igre Creativity večja od dimenzij vseh zaslonov naprav z Androidom, je temu potrebno prilagoditi način izrisovanja, saj vedno niso vidni vsi elementi igre.

Zato smo vse elemente igre shranili v tako imenovano zbirko (ang. collection), po kateri smo iterirali, za izris pa uporabili tako imenovani sistem kamere (ang. camera system), ki na podlagi svoje pozicije izrisuje elemente zbirke, s katerimi se prekriva. Če bi uporabnik želel videti drugo področje zaslona, bi se pozicija kamere premaknila na podlagi uporabnikovega premika po zaslonu. Kadar pa bi uporabnik želel videti večjo površino zaslona, bi se velikost in pozicija kamere povečala ali zmanjšala, elementi zbirke pa bi se pomanjšali.

Slika 20 prikazuje princip delovanja kamere, kjer je kamera označena s temno modro barvo, igralna površina pa s svetlo modro barvo. Uporabnik lahko kamero kadarkoli prestavi ali poveča preko zaslona na dotik.



Slika 20: Sistem kamere.

Vsi elementi zbirke so razširitev enega osnovnega razreda »Element«, ki vsebuje koordinate elementa (x,y,z), podatke o višini in širini, metodo za izris elementa na zaslon in metodo za posodobitev elementa. Elementi imajo dva dodatna atributa, ki določata, ali je objekt viden in ali je objekt statičen glede na sistem kamere.

Da na preprost način implementiramo sistem kamere, mora zbirka vsebovati metodo, ki na podlagi pravokotne površine in velikosti kamere vrne vse elemente zbirke, ki se prekrivajo s to površino. To omogoča preprosto implementacijo površine, po kateri se

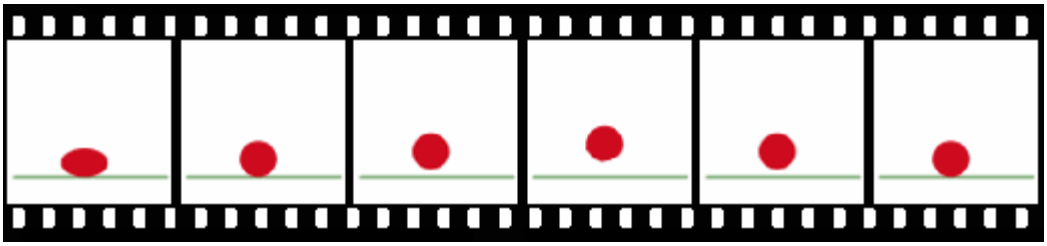
lahko uporabnik s pomočjo prstov premika in čemur sledi izrisovanje, saj izrisujemo le elemente, ki so v mejah prikaza kamere.

To metodo lahko uporabimo tudi za druge komponente igre, kot so na primer zaznavanje trkov elementov, zaznavanje pritiska na določen element in podobno. Ker igra Creativiy ne potrebuje zaznavanj trkov elementov igre, ta metoda služi le za izris elementov v mejah kamere.

4.8 Animacija elementov igre

Na tej stopnji je igra poleg logike dobila končno obliko, manjka le še animacija in premikanje elementov.

Animacija je iluzija gibanja, ki jo dosežemo s prikazovanjem različnih slik, ki se menjajo na podlagi časovne periode. Vsaka slika je nekoliko drugačna od predhodne, kar omogoča iluzijo. Animacije ponavadi delujejo s tako časovno periodo, da se slika zamenja 12-do 24-krat na sekundo [16].



Slika 21: Animacija poskočne žogice.

Slika 21 [16] prikazuje animacijo poskočne žoge, ki jo sestavlja šest različnih slik oziroma sekvenc animacije. Za prikaz animacije moramo definirati okvir, ki je enake velikosti kot posamezna slika. Okvir se potem s časovno periodo premakne za svojo dolžino in začne izrisovati naslednjo sliko oziroma sekvenco animacije.

Da bi v igro uspešno implementirali animacije, se moramo najprej odločiti, kako animacije shranjevati. Za vsak element lahko sekvenco animacije ločimo kot posamezno sliko, lahko pa uporabimo eno enotno sliko za element (ang. Sprite sheet), ki vsebuje vse sekvence in vse animacije, ki so potrebne za posamezen element. Če nimamo veliko animiranih elementov v igri, lahko za vse uporabimo eno enotno sliko, tako imenovano atlas (Slika 22 [18]). S tem vsi animirani elementi v igri uporabljajo eno samo fizično sliko.

Slika 22: Atlas animacij lika igre »The Prince of Persia«.

Ker igra Creativiy ne vsebuje veliko animiranih elemntov, smo se odločil za uporabo ene fizične slike oziroma atlasa za vse animacije vseh elementov v igri.

Ker animacija spreminja prikazano sliko s časovno periodo, lahko to delovanje implementiramo v metodi, kjer posodabljamo stanje igre. Ker ima osnovni element zbirke že vse potrebne metode, uporabimo kar njega.

Da se objekt lahko animira, mu moramo dodati naslednje atribute:

- trenutni okvir animacije,
- širino in višino okvirja,
- število okvirjov animacije in
- časovno enoto, na katero se mora slika zamenjati.

Slika 23 [17] prikazuje animacijo lika in oštevilčene okvirje animacije.

Slika 23: Animacija lika z oštevilčenimi okvirji animacije.

Naslednja slika (slika 24 [17]) prikazuje animacijo lika in premikanje okvirja animacije. Ko okvir pride do konca, se ustavi ali pa nadaljuje od začetka, odvisno od implementacije.

Slika 24: Premik okvirja animacije.

Metodi izrisa in posodobitve animiranega elementa po zaključku implementacije izgledata takole:

```
boolean animate = false;
boolean loopAnimation = false;
int currentFrame = 0;
int animationLength = 0;
int frameWidth = 0;
int skipTicks = 5;
int animationCount = 0;
Bitmap image = null;
Rect srcRect = new Rect(0,0,width,height);

void update() {
	if(animate) {
		this.animationCount++;
		if(this.animationCount == skipTicks) {
			this.currentFrame++;
			if(this.currentFrame > this.animationLength) {
				if(loopAnimation) {
					this.currentFrame = 0;
				} else {
					this.currentFrame = 0;
					animate = false;
				}
			}
			srcRect = new Rect(x+(this.currentFrame*this.frameWidth),y,width,height);
			this.animationCount=0;
		}
	}
}

void draw(Canvas c) {
	c.drawBitmap(image,srcRect,new Rect(x,y,width,height) ,null);
}
```

Metodi sta dokaj preprosti. Prva posodobi okvir animacije glede na časovni interval zamenjave slike, medtem ko druga izriše trenutni okvir animacije. Posodobitev stanja seveda upošteva, da se izvede 25-krat na sekundo.

4.9 Vpeljava hitrosti elementov

Da bi se elementi animirali v gibanju in ne samo na mestu, je potrebno implementirati tudi premikanje elementov na podlagi hitrosti. To dosežemo na enak način kot pri animacijah, upoštevati moramo le to, da se igra posodobi 25-krat na sekundo oziroma vsakih 40 ms. Da uporabnik ne bi opazil, da se stanje oziroma pozicija

elementa posodobi 25-krat na sekundo, je potrebno v metodo izrisa igre vpeljati interpolacijo, ki bo poskrbela, da se bo element bolj gladko premikal.

Funkciji v tem primeru izgledata tako:

```
boolean move = false;
boolean animate = false;
boolean loopAnimation = false;
int speedX = 0;
int speedY = 0;
int currentFrame = 0;
int animationLength = 0;
int frameWidth = 0;
int skipTicks = 5;
int animationCount = 0;
Bitmap image = null;
Rect srcRect = null;

void update() {
        if(animate) {
                this.animationCount++;
                if(this.animationCount == skipTicks) {
                        this.currentFrame++;
                if(this.currentFrame > this.animationLength) {
                                if(loopAnimation) {
                                        this.currentFrame = 0;
                                } else {
                                        this.currentFrame = 0;
                                        animate = false;
                                }
                        }
                        srcRect = new Rect(x+(this.currentFrame*this.frameWidth) ,y, width, height);
                        this.animationCount=0;
                }
        }
        if(move) {
                this.x = this.x + speedX;
                this.y = this.y + speedY;
        }
}

void draw(Canvas c, float interpolation) {
    c.drawBitmap(image,srcRect,new Rect(x+(speedX*interpolation),
y+(speedY*interpolation),width,height) ,null);
}
```

Metoda za izris je dobila nov argument, interpolacijo, ki pomaga pri boljši vizualni predstavi hitro premikajočih se elementov. Metoda za posodobitev stanja pa je dobila le posodobitve pozicije x in y glede na hitrost objekta.

4.10 Problemi pri izdelavi

Med razvojem ni bilo nobene resnejše težave, saj je na voljo dovolj dokumentacije, ki podrobno opiše razvoj igre. Eden večjih problemov se je pojavil le na emulatorju naprave, na kateri je tekel Android 1.6. Težava je bila v tem, da je sistem izpisoval opozorila zaradi prehitrega izrisovanja. Ker podobnih težav oziroma opozoril sistema nismo zasledili na fizičnih napravah, smo to težavo pripisali samemu emulatorju.

Poleg omenjenega problema smo naleteli tudi na manjšo težavo pri menjavi aktivnosti. Med menjavo dveh aktivnosti se je izvedla animacija, kar pa ni bilo zaželeno. To je bilo mogoče predelati, vendar je predelava odvisna od različice Androida - novejše različice ustrezne mehanizme in metode že vsebujejo, medtem ko mora v starejših različicah te spremembe izvesti razvijalec sam.

5 Sklepne ugotovitve

V diplomskem delu smo skušali predstaviti in se bolje spoznati z vsemi vidiki razvoja mobilne igre za platformo Android. Zaradi izkušenj, ki jih imamo pri razvoju tovrstnih aplikacij, predvsem tistih za platformo Java, smo se lotili tega projekta z velikim zanimanjem. Spoznali smo arhitekturo platforme in novosti, ki jih ta operacijski sistem prinaša. Predvsem smo želeli spoznati način delovanja sistema in pa načine reševanja nekaterih težav, s katerimi se soočajo razvijalci mobilnih vsebin.

Vse cilje, ki smo si jih zastavili smo tudi uspešno izpolnili. Igra, ki smo jo razvili, deluje konsistentno na vseh Android podprtih napravah in omogoča popoln izkoristek strojne opreme posameznih naprav.

Igra Creativiy je trenutno v zadnji oziroma končni fazi razvoja – testiranje. V kratkem času se bo pojavila na različnih trgih za mobilne vsebine.

Da bi predstavljena igra dosegla čim večji uspeh, bi jo bilo smiselno tudi prilagoditi za druge operacijske sisteme kot na primer I,OS.

Pri samem razvoju nismo naleteli na večje težave, kar je vsekakor rezultat dobre priprave in učinkovitega raziskovalnega dela. Izkušnje, ki smo jih pridobili, pa nam bodo vsekakor v veliko pomoč pri morebitnih novih projektih.

Kazalo slik

Literatura

[1] Mobile game.
Dostopno na: http://en.wikipedia.org/wiki/Mobile_game

[2] Snake is born: a mobile gaming classic.
Dostopno na: http://www.nokia.com/about-nokia/company/story-of-nokia/mobile-revolution/snake-game

[3] Mobile operating system.
Dostopno na: http://en.wikipedia.org/wiki/Mobile_operating_system

[4] Operacijski sistem.
Dostopno na: http://sl.wikipedia.org/wiki/Operacijski_sistem

[5] Java Platform, Micro Edition.
Dostopno na: http://en.wikipedia.org/wiki/Java_Platform,_Micro_Edition

[6] M. McShaffry, Game Coding Complete, Third Edition, Charles River Media, 2009

[7] Game loop.
Dostopno na: http://www.koonsolo.com/news/dewitters-gameloop/

[8] Koen Witters, deWiTTERS Game loop.
Dostopno na: http://www.koonsolo.com/news/dewitters-gameloop/

[9] Glenn Fiedler, Fix your timestep!
Dostopno na: http://gafferongames.com/game-physics/fix-your-timestep/

[10] Android operating system.
Dostopno na: http://en.wikipedia.org/wiki/Android_%28operating_system%29

[11] Android Developer's guide.
Dostopno na: http://developer.android.com/guide/index.html

[12] Android API reference.
Dostopno na: http://developer.android.com/reference/packages.html

[13] Eclipse software.
Dostopno na: http://en.wikipedia.org/wiki/Eclipse_%28software%29

[14] Writing Real-Time Games for Android.
Dostopno na:
http://www.google.com/events/io/2009/sessions/WritingRealTimeGamesAndroid.html
[15] Fixed-point arithmetic.
Dostopno na: http://en.wikipedia.org/wiki/Fixed-point_arithmetic

[16] Animacija.
Dostopno na: http://sl.wikipedia.org/wiki/Animacija

[17] Sprite Animation with Android.
Dostopno na: http://obviam.net/index.php/sprite-animation-with-android/

[18] Prince of Persia AtariAge forums.
Dostopno na: http://www.atariage.com/forums/topic/152500-prince-of-persia/